VHF & UHF ENERGY HARVESTING RADIO SYSTEM PHYSICAL AND MAC LAYER CONSIDERATIONS

by

XIAOHU ZHANG

B.S., Xi'an Jiaotong University, 2001

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2009

Approved by:

Major Professor
William B. Kuhn

Abstract

Wireless Sensor Network industrial and civilian applications have been moved closer to us since they were originally developed for defense applications. They have been or will be widely used in industrial process monitoring and control, earth quake monitoring, healthcare applications, construction health monitoring, home automation, traffic control, and space exploration. The IEEE802.15.4 standard defines the PHY and MAC layers for low power wireless sensor networks. However, applications and research of wireless sensor networking are centered on battery powered devices. To remove the battery from the system is the ultimate goal of this research by using Energy Harvesting technology, which will largely reduce the wireless sensor network maintenance cost, increase the option open to application environments and push the speed of wireless sensor network industrialization.

This thesis tackles the problem of RF link budget and PHY layer design for Energy Harvesting Wireless Sensor Network Nodes, through a *modification to PHY/MAC layers*. To this end, a prototype of energy harvesting radio is developed that hinges on burst-communication and solar cell energy harvesting techniques. The choice of operating frequency is considered relative to transmission range, antenna technology and RF link budget, and quantified by propagation measurements at four unlicensed frequencies in the VHF through UHF spectrums. A short preamble, PHY payload protocol frame structure and synchronization method are also proposed in order to support long sleep period duty cycle necessary in Energy Harvesting Radio systems. Some related work has recently begun under a standardization effort known as 802.15.4f. It is hoped that this thesis will contribute to this effort.

Table of Contents

List of Figures vii
List of Tables x
Acknowledgements xi
CHAPTER 1 Introduction 1
1.1 The Future of Wireless Sensor Networks 1
1.2 Current Research on Wireless Sensor Nodes 1
1.3 Challenges for Energy Harvesting Wireless Sensor Nodes 2
1.4 Thesis Organization 2
CHAPTER 2 K-State Energy Harvesting Radio (EHR) Prototype Demo Board 4
2.1 RF transmission mode Versus Power Consumption 5
2.2 Design Principle 6
2.3 Hardware Design 7
2.4 Software Design 11
2.4.1 RFIC Programming 11
2.4.2 Duty Cycle Design 13
2.4.2.1 High CPU Clock, Software Control TX 14
2.4.2.2 Low CPU Clock, Hardware Control TX 15
2.4.2.3 Computing Active and Sleep Time 16
2.5 Test Results 17
2.5.1 Lab Test 17
2.5.1.1 Current Consumption Test 17
2.5.1.2 The Burst Communication Test 18
2.5.1.3 The System Reliability Test 21
2.5.2 Outdoor Test 21
2.6 Conclusions 22
CHAPTER 3 RF Link Budgets of EHR Systems 23
3.1 RF Link Budget Factors 23
3.1.1 Propagation 24

3.1.2 Antennas 25
3.1.2.1 Reciprocity 25
3.1.2.2 Antenna Directivity 25
3.1.2.3 Gain 26
3.1.2.4 Effective Area 26
3.1.2.5 Antenna Size 26
3.1.3 Link Budget 27
3.1.3.1 Antenna Gain Effects on Link Budget 27
3.1.3.2 Link Budget Examples 28
3.1.4 RF Link Budget and The System Design of EHR 30
3.2 UHF/VHF Propagation Comparisons 30
3.2.1 Experimental Setup 31
3.2.1.1 Propagation Links 31
3.2.1.2 Experiment Environment 31
3.2.1.3 Antennas 34
3.2.1.4 Transmitter and Receiver 37
3.2.2 Experiment Results 39
3.2.2.1 Received Signal Strength Measurements at Each Frequency 39
3.2.2.2 Directional Antennas versus Monopole Antenna Results 42
3.2.2.3 Path Loss Exponent Comparison 46
3.2.2.4 Comparison of Four Frequency's Propagation 47
CHAPTER 4 Energy Harvesting Radio System PHY/MAC layer Considerations 49
4.1 IEEE802.15.4 & ZigBee Overview 50
4.1.1 PHY General Definition 50
4.1.2 MAC Layer Synchronization 51
4.2 Proposed EHR System PHY Layer 54
4.2.1 PHY General Definition 55
4.2.2 PHY Frame Structure 56
4.2.3 Preamble Design 57
4.2.3.1 Preamble Length versus Start of Frame Detecting Error 57
4.2.3.2 Method for Improving Short Preamble Frame Performance 60

4.2.3.3 Preamble Length versus Energy Comsumption 61
4.3 Proposed EHR MAC Layer Synchronization 62
4.3.1 Pure-EHR Network Obtaining Synchronization 63
4.3.2 Hybrid-EHR Networks for Obtaining Synchronization 66
4.3.3 Keeping Synchronization 69
4.4 Conclusions 69
CHAPTER 5 Conclusions 71
5.1 Summary 71
5.2 Challenges and Future Directions 72
Bibliography 73
Appedix A - PHY Layer implementation of K-State Energy Harvesting Receiver System 1
A.1 RFIC Front-end 1
A.2 DSP Baseband Design 4
A.2.1 Subsampling 4
A.2.2 Digital Low Pass Matched Filter 6
A.2.3 Bit-sync 9
A.3 DSP Software Implementation 12
A.3.1 Microcontroller Configurations 12
A.3.2 Software Architecture 14
A.3.3 Test Result 21
A.4 EHR DSP Code 25
A.4.1 ADC, LPF, and Bit-Sync File 25
A.4.2 PIC Initialization file 30
A.4.3 ISR Interrupt Process file 35
Appedix B - K-State Energy Harvesting Demo Board Schematic 37
Appedix C - Frequency Synthesis Board Schematic and Layout 39
Appedix D - K-State EHR Demo Board Code for 4MHz CPU Clock 41
Appedix E - K-State EHR Demo Board Code for 400kHz CPU Clock 51
Appedix F - Frequency Synthesizer PIC12F509 Code 61
F.1 151MHz Board 61
F.2 433MHz Board 66

F.3 902MHz Board 71
F.4 2400MHz Board 76

List of Figures

Figure 2-1 K-State Energy Harvesting Radio prototype board 5
Figure 2-2 RF operation mode (a) Continuous operation mode (b) Burst operation mode..... 6
Figure 2-3 K-State EHR demo board schematic..... 9
Figure 2-4 CASIO SA5511S4 solar cells and CASIO fx-260SLOAR Calculators..... 9
Figure 2-5 PIC16F676 Pin Diagrams 10
Figure 2-6 K-State Transceiver RFIC block diagram [10] 11
Figure 2-7 K-State RFIC programming register fields [10] 12
Figure 2-8 RFIC SPI Programming Timing sequence..... 13
Figure 2-9 1.1% Duty cycle with 4MHz CPU clock (above) times diagram (below)screen captures of RF output on scope..... 15
Figure 2-10 1.3% Duty Cycle with 400kHz CPU clock..... 16
Figure 2-11 1% duty cycly capacitor charge-discharge process..... 17
Figure 2-12 Microcontroller PIC16F676 Current consumption measurement with 4MHz clock (a) ACTIVE mode current (b) SLEEP mode current..... 18
Figure 2-13 Four burst clusters with 1% duty cycle 19
Figure 2-14 One Active period command signal and burst signal of 400kHz Clock Speed design 19
Figure 2-15 Five burst period 20
Figure 2-16 433.92MHz modulation signal..... 21
Figure 2-17 Yaesu VR-120 Receiver..... 21
Figure 2-18 EH demo board transmitting range test (a) Satellite view of 0.2km transmission range (b) view look back from parking lot (c) view looking inside hallway of RA2097 [11] 22
Figure 3-1 Calculated range at 433 MHz with dipole antennas and 1 kbps data rate..... 29
Figure 3-2 Calculated range at 2.4 GHz with dipole antennas and 1 kbps data rate 29
Figure 3-3 Two type of experimental Propagation Links..... 31
Figure 3-4 Rathbone Hall Engineering Building constructions..... 32

Figure 3-5 Measurements location marked on the indoor floor plan and outdoor map : Left is 2nd floor plan of Rathbone Hall; Right is the Google earth map of Rathbone Hall 33
Figure 3-6 Two path used for lower frequency transmitting range measurement 33
Figure 3-7 Antennas used in measurement 34
Figure 3-8 Monopole antenna, Ground plane and antenna reflection coefficients measurements 35
Figure 3-9 151MHz directional antenna S11=-14.9dB, monopole antenna S11=-12.7dB 36
Figure 3-10 433MHz directional antenna S11=-28dB, monopole antenna S11=-12.9dB 36
Figure 3-11 902MHz directional antenna S11=-12.5dB, monopole antenna S11=-10dB 36
Figure 3-12 2400MHz directional antenna S11=-31dB, monopole antenna S11=-13.3dB 37
Figure 3-13 Portable 10mW Transmitter photo 37
Figure 3-14 Receiver—Spectrum and monopole antenna 39
Figure 3-15 151MHz two different link propagation with Path loss Exponent 43
Figure 3-16 433MHz two different link propagation with Path loss Exponent 44
Figure 3-17 902MHz two different link propagation with Path loss Exponent 45
Figure 3-18 2400MHz two different link propagation with Path loss Exponent 46
Figure 3-19 Four frequencies propagation comparison 47
Figure 4-1 K-State Micro-Transceiver Demo Board 50
Figure 4-2 Frequency band and Data rate of IEEE802.15.4 (IEEE802.15.4 2006) 51
Figure 4-3 ZigBee beacon-enabled TX / RX mode [1] 51
Figure 4-4 IEEE 802.15.4 Superframe structure [1] 52
Figure 4-5 IEEE802.15.4 super frame sequence with different Beacon Oder 52
Figure 4-6 IEEE802.15.4 Sync Time Vs. Duty Cycle 54
Figure 4-7 PHY Frame Format 56
Figure 4-8 Preamble test experiment environment 58
Figure 4-9 Error Probability of SOF versus Input Signal Power of varied preamble length 59
Figure 4-10 SOF detection with various preamble length 60
Figure 4-11 Energy Consumption VS. Preamble Length 62
Figure 4-12 Beacon Frame format 63
Figure 4-13 Beacon searching process 64
Figure 4-14 Beacon detected process 64
Figure 4-15 Energy Harvesting Radio Sync time versus Duty cycle 66

Figure 4-16 Hybrid-EHR System Synchronization 68
Figure 5-1 IF filter and wideband FM modulation 2
Figure 5-2 FSK demodulation (a) Correlator Receiver (b) Non-correlator Receiver 2
Figure 5-3 One-shot FSK demodulation: (a) Modulation Data, (b) FM Modulated waveform, (c) Zero cross Sampling clock, (d) Zero cross pulse & average energy curve 3
Figure 5-4 Subsampling 10.7 MHz with 75kHz sampling rate 5
Figure 5-5 Two pole IIR digital low pass filter 6
Figure 5-6 Frequency response comparison of designed 1 kHz Low Pass Matched Filter with MATLAB function plot 8
Figure 5-7 output of LPMF (a) Ideal output (b) real output 9
Figure 5-8 Bit-sync data versus clock 10
Figure 5-9 Bit-sync Finite State Machine (FSM) 11
Figure 5-10 K-State Micro-Transceiver Demo board block diagram 12
Figure 5-11 FSK Demodulation DSP software flow 15
Figure 5-12 Timer2 ISR 16
Figure 5-13 LPMF procedure 17
Figure 5-14 LPF calculation coefficient *b* products 19
Figure 5-15 LPF calculation coefficients a products 20
Figure 5-16 Calculated X shifts 21
Figure 5-17 PHY Layer DSP test enviroment 21
Figure 5-18 10kHz Bandwith FSK modulation 22
Figure 5-19 "Zero-cross" detection 22
Figure 5-20 512bps Bit-sync Test restults 23
Figure 5-21 512bps Bit-sync Test restults 24
Figure 5-22 Schematic 37
Figure 5-23 Layout 38
Figure 5-24 Schematic 39
Figure 5-25 Layout 40

List of Tables

Table 2-1 Demo board Electrical Specification 7
Table 3-1 quarter-wave antenna size and effective aperture of dipole antenna 34
Table 3-2 Components list of 10mW un-modulated transmitter 38
Table 3-3 151MHz Propagation measurement results 39
Table 3-4 433MHz Propagation measurement results 40
Table 3-5 902MHz Propagation measurement results 41
Table 3-6 2400MHz Propagation measurement results 41
Table 3-7 Path Loss Exponents comparison 46
Table 4-1 IEEE802.15.4 Synchronization time calculation 53
Table 4-2 Barker code 56
Table 4-3 Preamble Length VS. Energy when data rate is 512bps 61
Table 4-4 Sync time and power consumption comparison of EHR and IEEE802.15.4 standard radio 65
Table A-1 Two pole LPF coefficients 7
Table A-2 Bit shift and product factor 18
Table A-3 LPF coefficients 2^n expression 18

Acknowledgements

This thesis is obviously not the product of my own individual efforts, but the fruit of intense collaboration. First and the foremost, I would like to thank my major advisor, Professor Bill Kuhn for his excellent guidance, above and beyond the mere technical research. He has truly given meaning to the word “advisor”, helping me every step of the academic way. In addition, I would also like to express my gratitude towards the other members of my committee, Dr. Don Gruenbacher and Dr. Bala Natarajan, for their support and feedback, and my colleagues in Wireless Communication Hardware design group.

I also gratefully acknowledge our research sponsor – Peregrine Semiconductor for their support.

CHAPTER 1 Introduction

1.1 The Future of Wireless Sensor Networks

Wireless sensor networks (WSN) and associated wireless sensor node devices in industrial and civilian applications have seen rapid growth since they were originally motivated by military applications. They have been or will be widely used in industrial process monitoring and controls, earthquake monitoring, healthcare applications, construction health monitoring, and home automation and traffic control. Additionally, they will help humans to explore outer space, planets, and moons. For example, future Mars rovers could plant hundreds of WSN nodes on the planet during exploration and then collect data from all nodes over time. They will help humans to inspect and monitor the south and north poles, deserts, virgin forests, unfathomable depths of the oceans and no man's lands on the earth. To implement all these applications, *battery-free* and *Energy Harvesting Radio(EHR)* nodes are preferred. Energy harvesting nodes use energy captured from natural or human environments, such as solar, thermal, wind, biological and kinetic processes. Unlike traditional battery powered systems, the advantages of energy harvesting devices reduce or eliminated the need for manual intervention. Moreover, waste from depleted batteries is eliminated, many application environment restrictions are removed, and the WSN lifecycle is extended from months or years, to decades or centuries.

1.2 Current Research on Wireless Sensor Nodes

The wireless sensor node is an important part of a wireless sensor network. It must handle the tasks of gathering sensory information, processing data, and communicating with the network. For designing current wireless sensor nodes, IEEE802.15.4 is a main standard to interface with.

IEEE Std 802.15.4 defines the physical layer (PHY) and medium access control (MAC) sublayer specifications for low-data-rate wireless connectivity with fixed, portable, and moving devices with limited battery consumption, typically operating in the personal operating space (POS) of 10m [1]. In the definition, three main frequency bands are used, 2.4GHz for global, 915MHz in America and 868MHz in Europe. The bit rate ranges from 20kbps to 250kbps. BPSK, ASK, and O-QPSK may be used, with DSSS, OCDM (Orthogonal Code Division

Multiplexing) spreading methods at the physical layer. CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance) channel access method is used at the MAC layer.

Most current wireless sensor nodes research is focused on battery consumption improvements [2] [3] [4], energy/power management [5], application of wireless sensor nodes [6], and energy harvesting methods [7]. Few are proposing a PHY or MAC layer. Research of narrow-band, spread spectrum, and ultra-wideband (UWB) technologies of PHY layer has been evaluated [8]. Research of MAC layer has resulted in talk of the mobility of WSN [9]. Few self-powered wireless radio commercial products were developed [24]. However, almost all of them assume using IEEE802.15.4, Blutooth, or UWB as PHY layer construction and *few* have proposed approaches to *optimize PHY and MAC sublayer on a system level* (although very recently a new 802.15.4f standards proposal has been launched [25]).

1.3 Challenges for Energy Harvesting Wireless Sensor Nodes

The challenges of energy harvesting wireless sensor nodes include several aspects: developing an efficient energy harvester, implementing suitable radio systems, and providing high energy storage capability. The focus of this thesis is on the radio systems challenges and associated solutions. The radio system design challenge includes:

1. *RF transmitting mode versus power consumption*
2. *Low power circuit design*
3. *Frequency selection versus transmitting performance*
4. *Antenna technology versus RF link budget*
5. *PHY layer frame construction versus energy consumptions*
6. *Ultra-low duty cycle communication synchronizations*

1.4 Thesis Organization

In this thesis, three chapters address the energy harvesting wireless sensor nodes design challenges listed above. Chapter two addresses points 1 and 2. An EHR prototype demo experiment shows a burst communication system employing common indoor solar energy harvester panels and capacitor energy storage, and discusses methods for low power circuit design. Chapter three is focused on points 3 and 4, addressing *frequency selection versus transmitting performance* and *antenna technology versus RF link budget* considerations. Four

frequencies' propagation performance were measured in the VHF/UHF frequency band including both indoor and outdoor enviroments. Chapter four focuses on points 5 and 6 and investigates the PHY frame construction, and synchronization problems. Finally chapter five (Conclusions) summarizes overall recommendations and possible future directions for the research area.

CHAPTER 2 K-State Energy Harvesting Radio (EHR) Prototype Demo Board

In this chapter, the design and testing of K-State's EHR prototype demo board (Figure2-1) are discussed. The *purpose of this part is to prove feasibility of an energy harvesting radio system* by using burst communication mode techniques at 433MHz operation frequency and currently available radio technology. The motivation of this is that most current research and industrial products of wireless sensor nodes are still using batteries as power supply. They are focused on ways to improve energy management methods, energy efficient wireless sensor network protocols, or low power SoC (System on Chip) technologies for short-range use. Unlike these efforts, K-State's EHR prototype is a battery-less wireless sensor node demonstration. The demo shows that the energy harvested by four $5cm^2$ solar panels from indoor lighting is enough to support 433MHz bust mode wireless communication with range approaching 0.2km or more. In this system, the K-State microtransceiver RFIC [10] is used as the RF front end, and solar cells from low-cost calculator products are used as the energy source. A 16F series PIC micro-controller serves as the data source and timing subsystem.

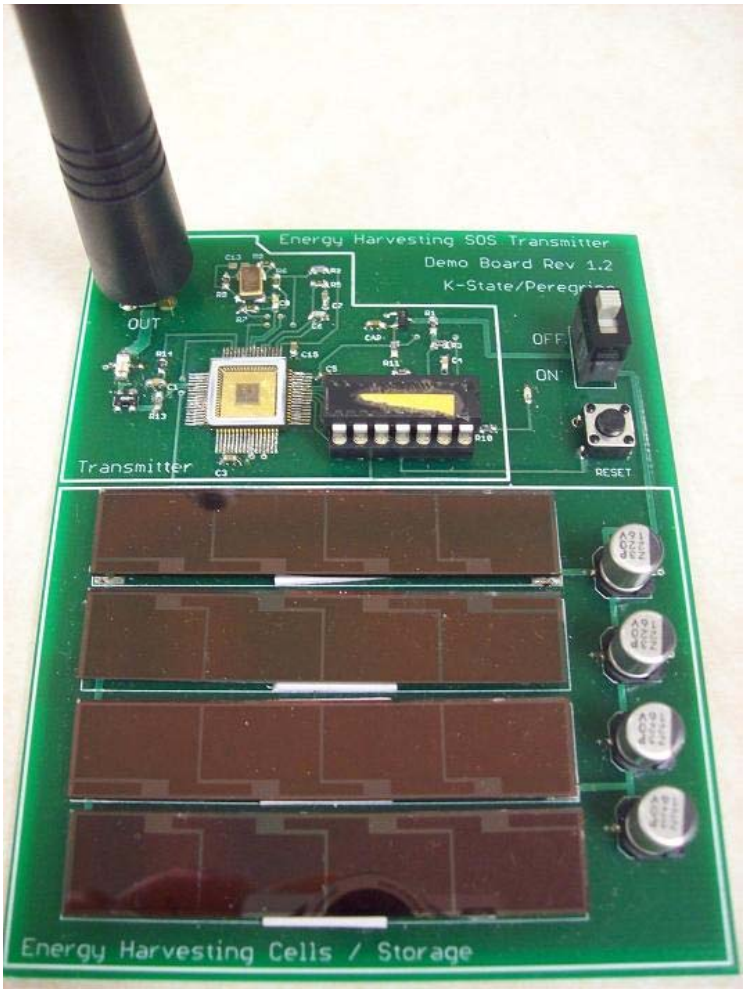


Figure 2-1 K-State Energy Harvesting Radio prototype board

2.1 RF transmission mode Versus Power Consumption

For an EHR system, to keep the average power consumption at the micro-watt levels available with indoor solar-cell harvesters, we must reduce both the average digital baseband power consumption and the average RF transceiver power consumption. For a digital baseband circuit, the power consumption scales down when processing clock speed reduces, such as the DSP chip operation clock. Thus, low power systems can be implemented when processor clocks down to the *kHz* range and continuous operation is possible. Taking notice, the data rate will be reduced respectively. Unfortunately, RF circuit power consumption does not reduce when the data rate goes lower. The analog parts in RF circuits, such as LNA and VCO must still consume the same current even if data rate is lowered. Many techniques are being researched to design low power RF circuits, such as using very high quality passive elements, Q-enhanced LC

resonant or MEMs circuits to replace the preselect filter and LNA, and using lower power IC processes, etc [11], but to-date, none of these techniques offers the possibility of micro-watt RF transceiver functions.

Fortunately, *burst mode* operation is an effective way to reduce overall average power consumption. In burst mode, the active operation period ***T*** is broken down into several pieces or time slots, $t_1, t_2, \ldots t_n$ including *active subperiods* and *sleep subperiods* where $T = t_1 + t_2 + \ldots + t_n$. Thus the whole transmission data volume is broken down into several parts, $d_1, d_2, \ldots d_n$, . Figure2-2 shows the comparison of continuous and burst modes. The red symbol P_{AVG} and the dash lines are indicating the average power consumption of operating period. Obviously, using burst mode saves power more than using continuous mode.

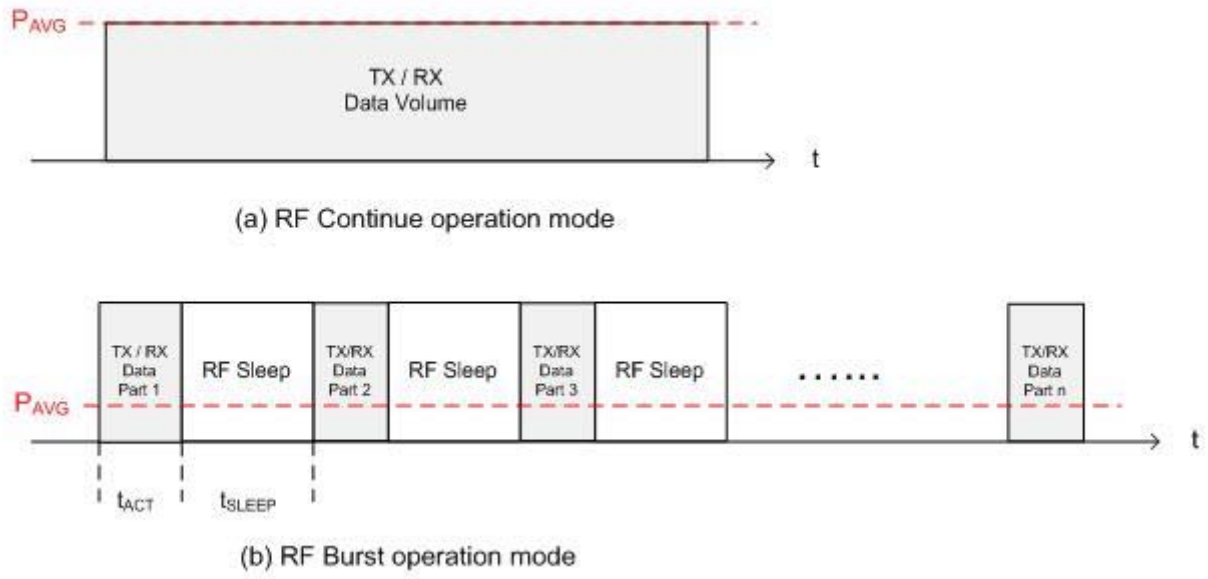


Figure 2-2 RF operation mode (a) Continuous operation mode (b) Burst operation mode

Therefore, the burst mode has significant advantage for EHR systems since there *active and sleep alternated processes* allow the energy to be re-charged during the SLEEP periods and discharged during the ACTIVE periods.

2.2 Design Principle

The burst communication mode is the fundamental design idea of the EHR demo board. To simulate a data burst, we modulated a 433.92MHz sine-wave signal by a 1kHz square wave

burst. Using 1kHz square burst is the same as 1kHz amplitude-shift-keying (ASK) modulated RF signal. With the simulated 1010 pattern created with a 0.5ms square wave burst, a human ear recognizable "beep" tone can easily be detected by a commercial receiver for demonstration and range testing purposes.

The AM square wave burst envelope was generated in two ways: (1) Hardware control method : turn on and off the radio frequency chip's RFIC_PW_DN pin directly at 1kHz frequency; (2) Software control method : program "LPAena" bit of the RFIC control register to switch radio frequency chip between transmitting and standby modes. These two ways were designed for different system configurations of CPU operating clock speed and energy storage capacitor's value. The first method, by controlling RFIC hardware pin, requires less CPU instruction cycles because of small code size. By contrast, the second method, by programming the RFIC register for each pulsing, requires more CPU instruction cycle resources since code size is bigger.

In the design of the whole system, these two methods could be integrated to exploit the tradeoffs between the energy consumption and timing consumption. In later parts of this chapter, these two different design and measurement results are discussed and compared.

2.3 Hardware Design

The main components of the EHR demo board hardware included a K-State RFIC, a commercial 5 mW TCXO, a Microchip PIC16F676, a voltage regulator, and SMT Capacitors. Table 2-1 is the EHR board's electrical specification.

Table 2-1 Demo board Electrical Specification

Duty cycle	≤ 1.3%
solar cell output Voltage (two in series)	5.4v
Solar cell output Current (indoor fluorescent lights)	0.02mA
Solar cell output Current (40-Watt incandescent bulb with 6 inches distance)	0.32mA
RFIC supply Voltage range	3.0v~3.3v
RFIC TX mode Current	≤20mA
RFIC Sleep mode Current	≤0.01mA

RFIC TX Frequency	433.92MHz
PIC16F676 Operating mode Voltage	1.2v~3.0v
PIC16F676 Operating mode Current (4kHz)	≤0.5mA
PIC16F676 Operating mode Current (4MHz)	≤0.74mA
PIC16F676 Sleeping mode Current(4kHz)	≤0.05mA
PIC16F676 Sleeping mode Current(4MHz)	≤0.17mA
3.3v Voltage Regulator Input Voltage range	3.7v~12v
3.3v Voltage Regulator Output Voltage range	3.3v +/-5%
3.3v Voltage Regulator Gnd current	≤0.5mA

$$I_{avg} = \frac{(I_{PIC\_act} + I_{RF\_act}) \times T_{act} + (I_{PIC\_sleep} + I_{RF\_sleep}) \times T_{sleep}}{T_{act} + T_{sleep}} \quad (2\text{-}1)$$

Depends on the electrical specification, we can get the average current of the system using the equation (2-1). In the equation (2-1), $\boldsymbol{I_{avg}}$ is average current of system, $\boldsymbol{I_{PIC\_act}}$ is the current of the Microcontroller consumed during the active period, $\boldsymbol{I_{RF\_act}}$ is the current of the RFIC consumed during the active operation period, $\boldsymbol{I_{PIC\_sleep}}$ is the current of the Microcontroller consumed during the sleep mode, $\boldsymbol{I_{RF\_sleep}}$ is the current of the RFIC consumes during the sleep mode, $\boldsymbol{T_{act}}$ is total active time slots and $\boldsymbol{T_{sleep}}$ is total sleep time slots. The duty cycle, which is the ratio of $\frac{T_{act}}{T_{act} + T_{sleep}}$, can be designed depending on the system requirements, system power consumption, energy harvesting capability, and energy storage capability. In section 2, 4, example calculations show the relationship of the duty cycle, average current and voltage variation.

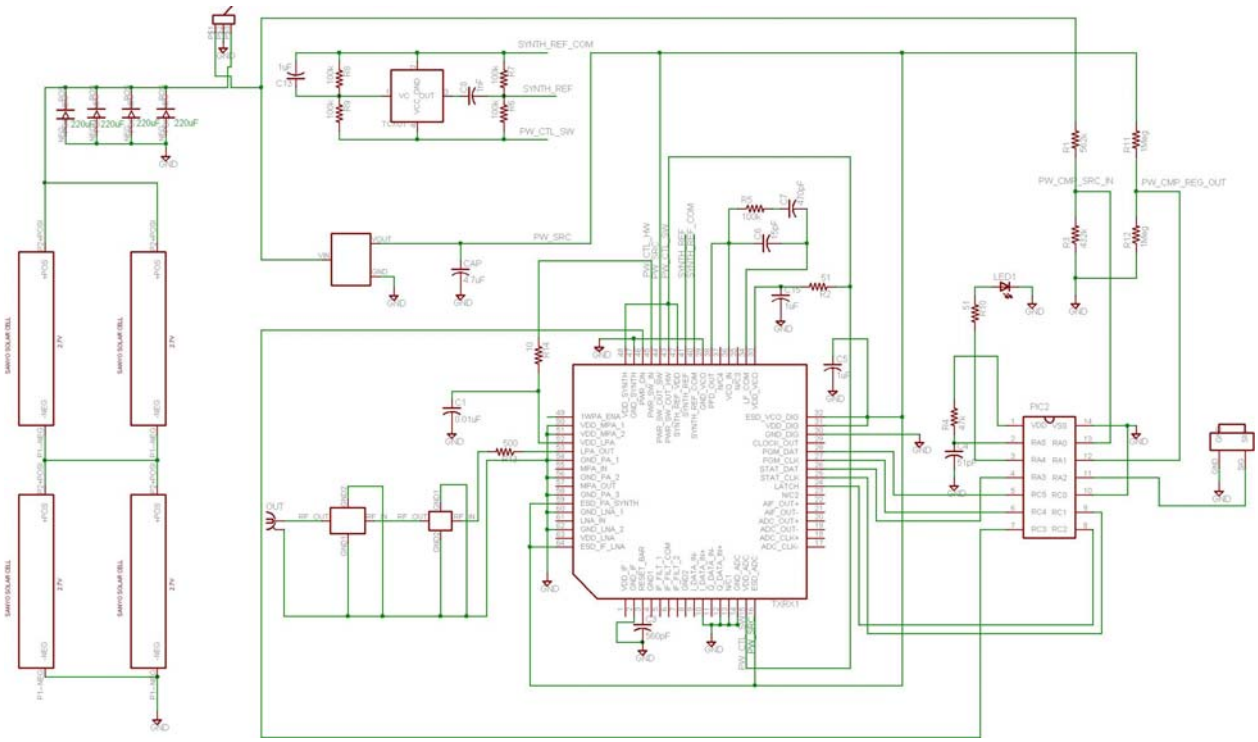


Figure 2-3 K-State EHR demo board schematic

Figure 2-3 is the K-State EHR demo board schematics. The layout is in Figure A-2. The energy is harvested by using four CASIO SA5511S4 solar cells (Figure 2-4). Each can provide 2.7v dc-voltage, 0.16mA dc-current under a 40-Watt incandescent bulb with 6 inches distance. Otherwise, each can provide 2.7v dc-voltage, 0.033mA dc-current under indoor fluorescent lights. These solar cells were salvaged from CASIO fx-260SLOAR commercial Calculators. To maximize energy harvesting capability, two of them were attached in series and then parallel led with another series two to provide 5.4v output at higher current.

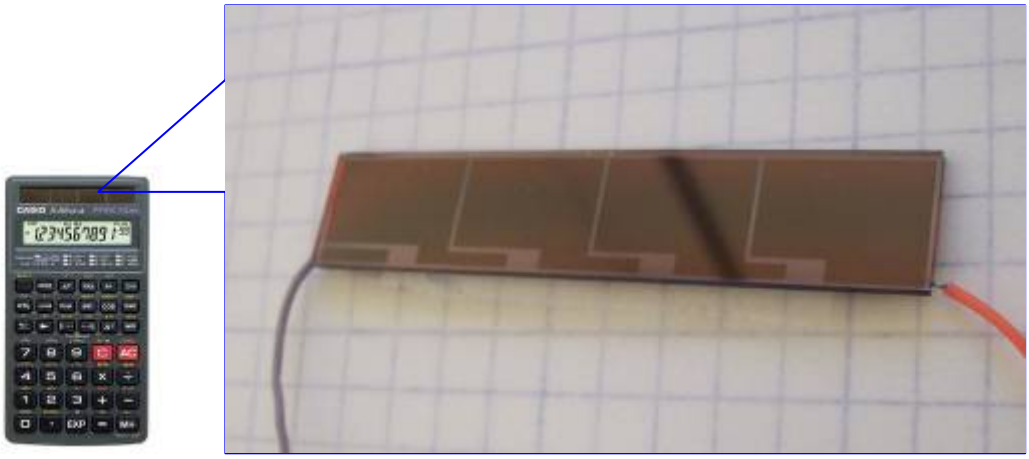


Figure 2-4 CASIO SA5511S4 solar cells and CASIO fx-260SLOAR Calculators

The energy storage is using four 220uF SMT capacitors, which are connected in shunt between solar cells and power switch. A TPS76130-100MA voltage regulator is put after the power switch to provide stable 3.3v dc-voltage for the RFIC and the Microchip PIC.

The 14 pins chip in the schematic is a Microchip PIC16F676, which is a Flash-Based 8-bit CMOS, high performance, low power, and wide operating voltage range (2.0v~5.5v) RISC CPU. Figure 2-5 shows PIC16F676 pin diagram. This microcontroller can be configured to work with several different clock speeds, either an external or an internal clock oscillator.

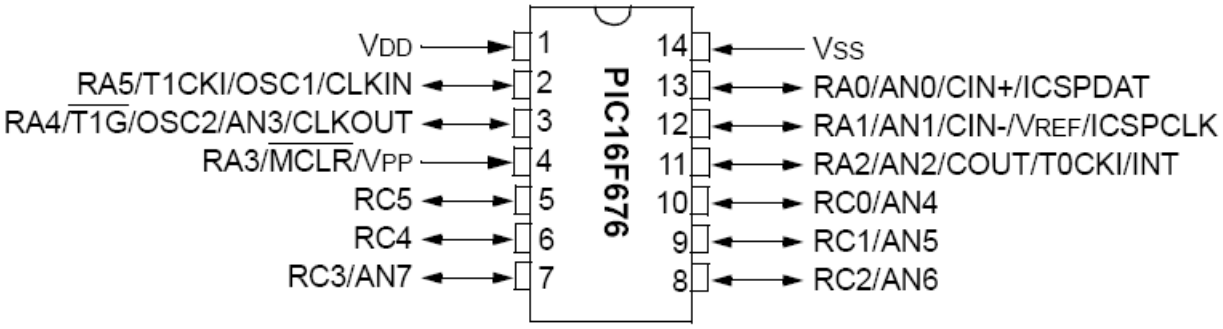


Figure 2-5 PIC16F676 Pin Diagrams

The biggest chip in the schematic is K-State RFIC. Figure2-6 shows the RFIC block diagram. The *PgmClk*, *PgmData* and *PgmLatch* are serial port interface signals defined by the RFIC. The IF_{out} and IF_{in} are the 10.7MHz Intermediate Frequency output and input, which connect with an off-chip IF filter. The *ADCclk* is 1bit ADC sampling clock input pin. The *ADCout* is the 1bit ADC sampling data output pin. The *Ref* is a 19.2MHz TCXO reference clock input pin.

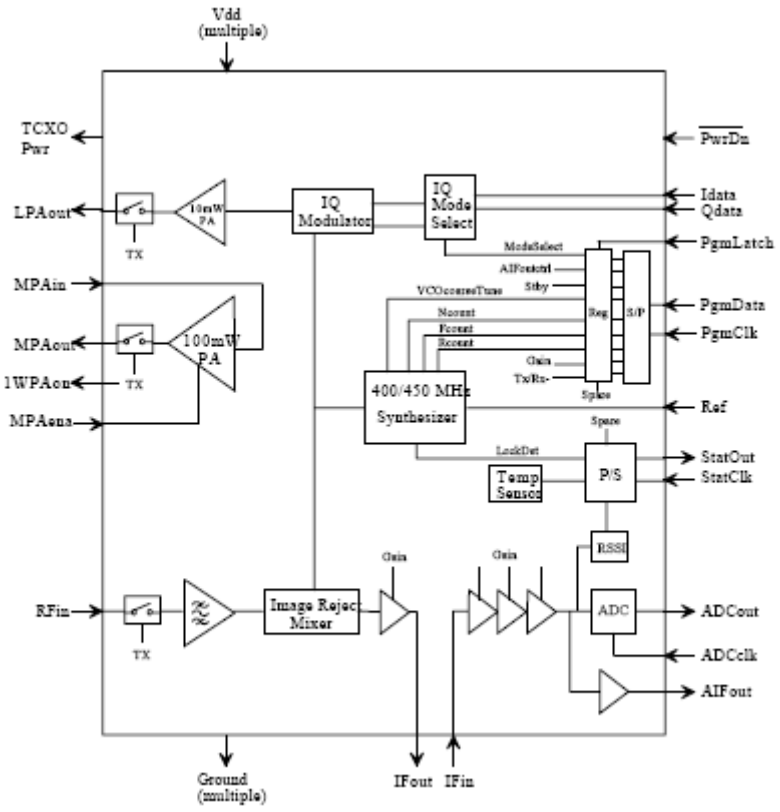


Figure 2-6 K-State Transceiver RFIC block diagram [10]

A 19.2MHz TCXO oscillator is used as RF reference clock, which consume 5mW (1.5mA at 3.3v). A DC-490MHz low pass filter is put between the attenuator and the antenna for transmitter harmonic attenuation in and receiver preselection/protection from cell phone and other strong nearby emitters.

2.4 Software Design

2.4.1 RFIC Programming

The first function of the software is programming the RFIC. The RFIC can be programmed to work in TX, RX, or SLEEP modes through a serial port interface (SPI). The Microcontroller programs the registers, which are inside the RFIC, through the SPI. Figure2-7 shows the RFIC programmable register fields [10].

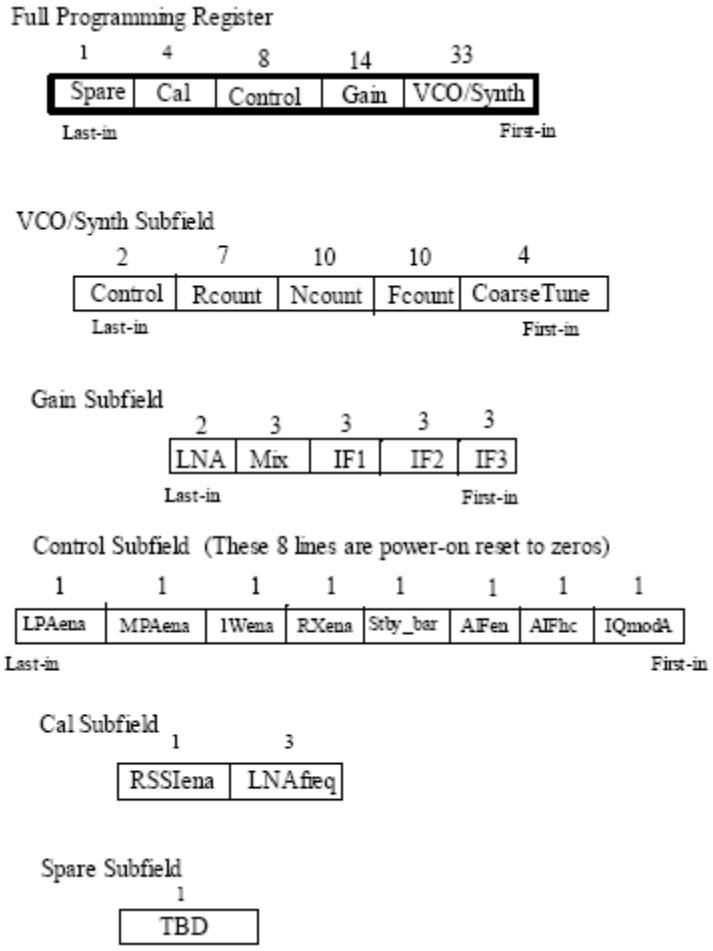


Figure 2-7 K-State RFIC programming register fields [10]

There are 60 bits in the programmable register, which are divided into four main function subfields. The VCO/Synth subfield is used to set the synthesizer. The Gain subfield is used to control low noise amplifier and IF amplifier. The Control subfield is used to control the chips working mode. This includes enable and disable power amplifier, RX/TX mode switch, Sleep/ Standby mode switch, etc. The Cal subfield is used to configure RSSI and set the LNA frequency. To configure the VCO/Synth, equation (2-2) is used to calculate N and R. Here f_{ref} is the reference frequency, which is 19.2MHz, and f_{vco} is the frequency of the signal generated by

$$\frac{f_{ref}}{R} = \frac{f_{vco}}{N} \tag{2-2}$$

VCO. In "A Low-Power, Radiation-tolerant, RFIC Micro-Transceiver Chipset for Space Application" [10], detailed programming steps for the RFIC are provided. Programming should begin with *Pgm_dat*, *Pgm_clk*, and *Latch* input pins at zero. Place first bit onto *Pgm_dat* input

pin. Then bring *Pgm_clk* high and return it to low. Repeat with subsequent bits until all 60 bits are entered. Finally, raise *Latch* line to high state and then issue one more clock to latch data into the chip. No changes occur in the RFIC chip's operation until this final clock rising edge with Latch is asserted [10]. Using this description, the programming time sequence is designed as shown in Figure 2-8.

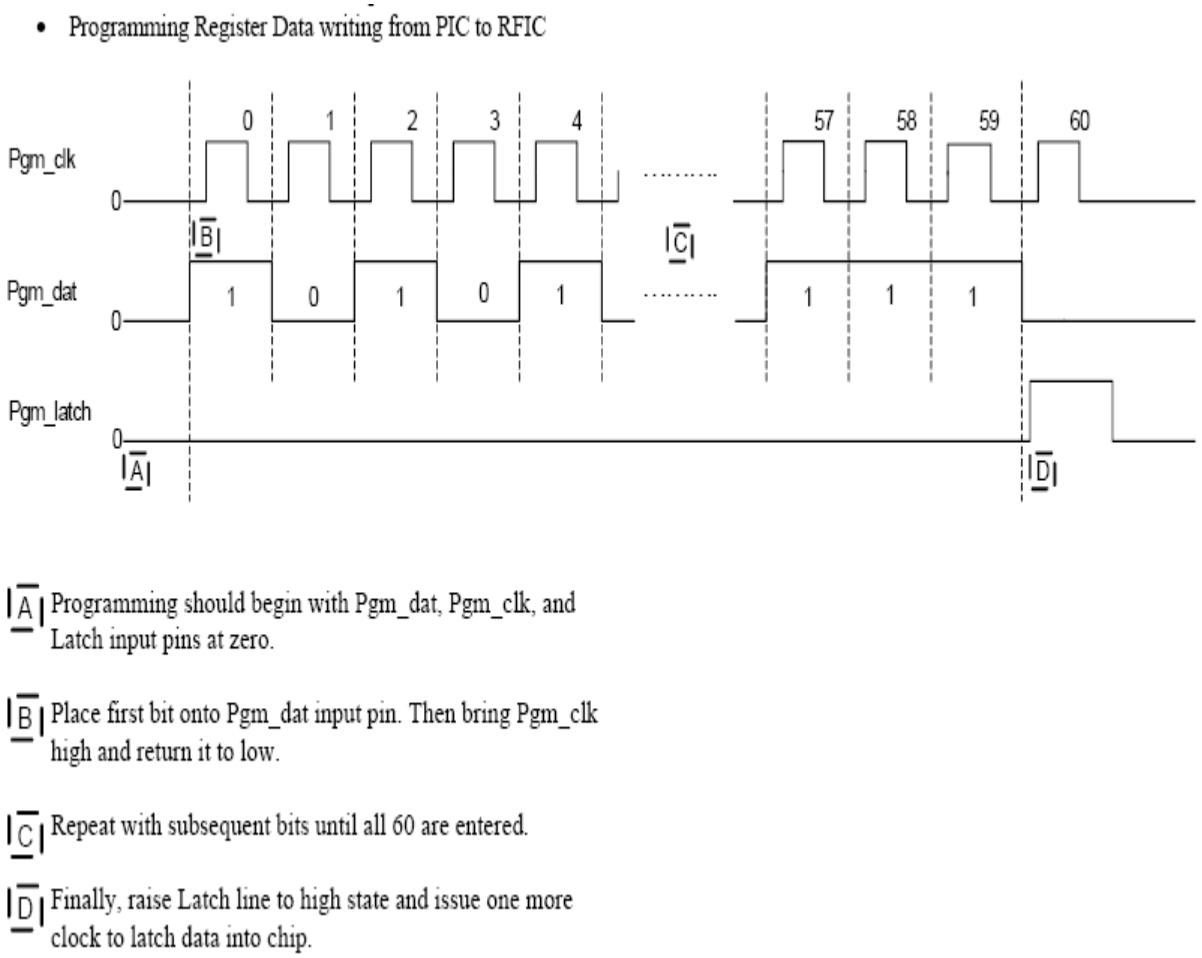


Figure 2-8 RFIC SPI Programming Timing sequence

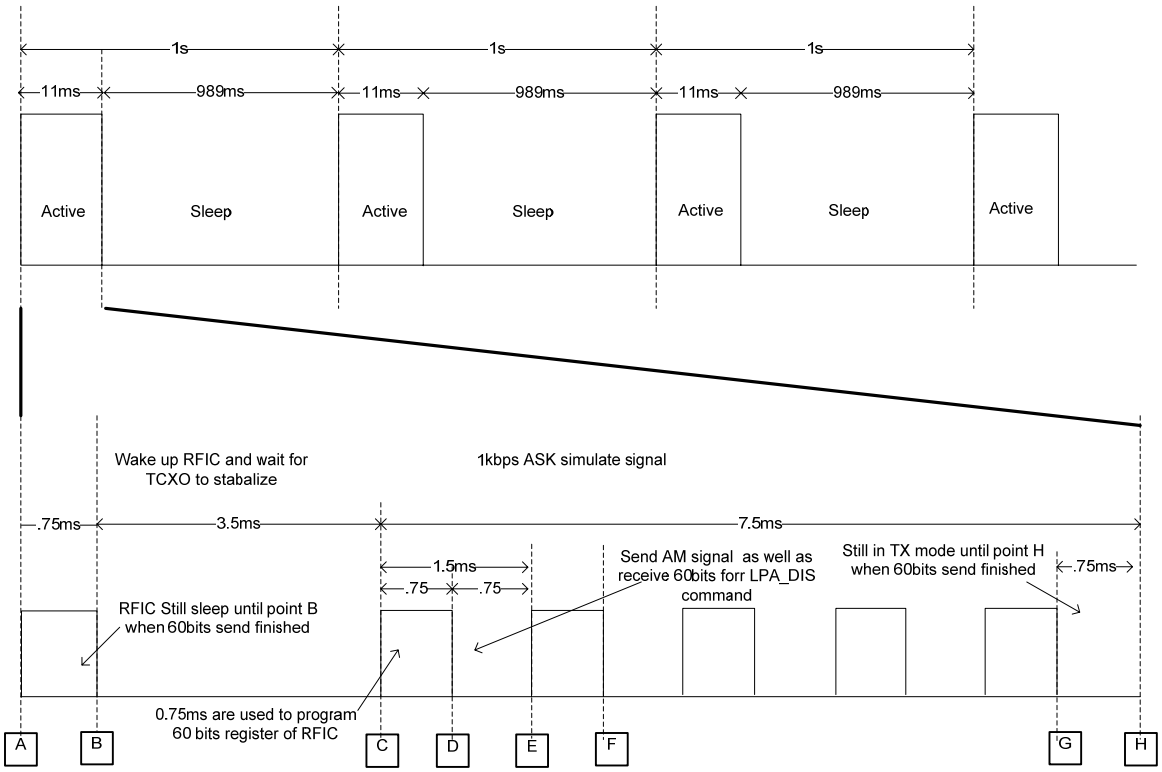
2.4.2 Duty Cycle Design

The second main function of the software is the duty cycle timing control. The software controls the time period switching of the RFIC transmission, sleeping, and energy recharging. Timing is critical to keep the system working in a low energy environment. To design the critical timing, we should know the accurate CPU instruction operation cycle time. For the PIC16F676, *one instructions cycle consists of four oscillator periods*. As mentioned in the previous design principle part, there are two different ways used to implement the RFIC control: one is software

controlled and the other one is hardware controlled. Using the software controlled method; the CPU oscillator clock is configured with 4 MHz internal high CPU oscillator speeds, which has a *1 us instruction cycle*. Using the hardware controlled method; the clock speed can be reduced to conserve energy. In this case, the CPU oscillator clock was configured with 400kHz external RC low CPU oscillator speeds, which has a *10 us instruction cycle*. These two different configurations implementations are discussed and compared below.

2.4.2.1 High CPU Clock, Software Control TX

In this design, the microcontroller PIC uses a 4MHz system clock configured by using the internal oscillator. The high clock speed is needed in the software controlled case to allow repeated program of the 60 bits control register to modulate the bits. The code was written in assemble language and the total programming time of writing 60 bits RFIC registers is 0.75ms. The code is in Appendix-D. Figure 2-9 shows the resulting 1.1% duty cycle design software processing sequence diagram and the RF output captured on an oscilloscope.



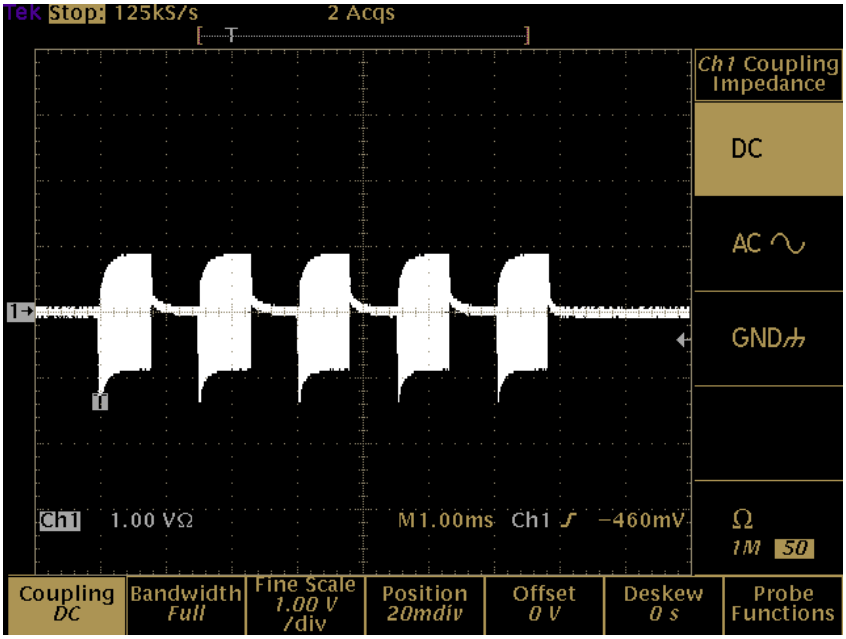


Figure 2-9 1.1% Duty cycle with 4MHz CPU clock (above) times diagram (below)screen captures of RF output on scope

According to equation (2-1), the average current of this design is $0.41mA$.

$$I_{avg} = \frac{(0.74mA + 20mA) \times 11ms + (0.17mA + 0.01mA) \times 989ms}{1000ms} = 0.4062mA$$

From above calculated result, it is easy to estimate approximate average power consumption of the EHR demo board, which is 0.4062mA*3.3v=1.34mW. To compare the result with using batteries instead of the solar cells for this radio, assume we are using a 2.7Ah AA batterie. This battery could support the radio board for about 6650 hours (about 277 days). In other words, the battery needs to be changed in under a year, whereas the HER radio could function indefinitely.

2.4.2.2 Low CPU Clock, Hardware Control TX

The second configuration uses a lower CPU clock. The duty cycle is designed as 1.3% and the PIC operating clock is using an external RC 400kHz oscillator. Figure 2-10 shows the software processing diagram.

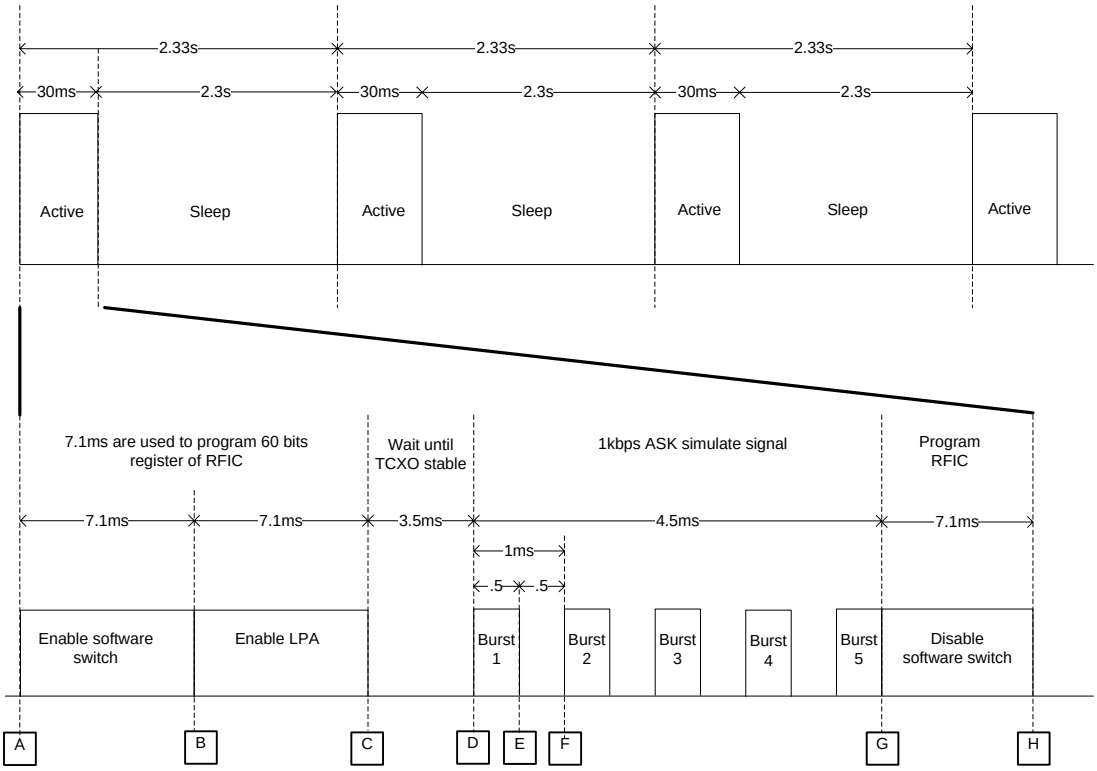


Figure 2-10 1.3% Duty Cycle with 400kHz CPU clock

2.4.2.3 Computing Active and Sleep Time

From Figure 2-10, we can find the active time to be 30ms and the sleep time is 2.3s. To determine this duty cycle, we must determine the charging and discharging time of the capacitors. The equation (2-3) describes the relationship of the voltage, current, and the capacitor values where ***dv/dt*** is the voltage changing or discharging rate, ***i*** is the current, and ***C*** is the capacitor value.

$$i = \frac{dv}{dt}C \tag{2-3}$$

From section 2.3, we note that the EHR demo system has four parallel 220uF capacitors, the microcontroller PIC active current is 0.5mA, and the RFIC transmitting current is 20mA. Thus the voltage dischargs from capacitor during the 30ms active period is given:

$$dv = \frac{i \times dt}{C} = \frac{(0.5 + 20)mA \times 30ms}{4 \times 220uF} = 0.7v$$

The time to recharge this 0.7v voltage back using four solar cells, which provides 2x0.13mA current, is:

$$dt = \frac{dv}{i} C = \frac{0.7v}{2 \times 0.13mA} \times 4 \times 220uF = 2.3s$$

This is why the sleep time was designed as 2.3s when the active time is 30ms. Figure 2-11 shows the capacitors charging and discharging process.

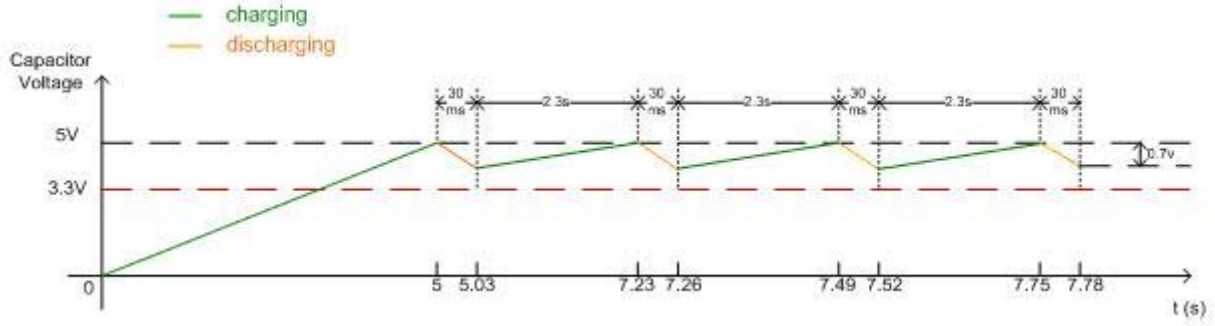


Figure 2-11 1% duty cycly capacitor charge-discharge process

The average current of this design can be calculated from equation (2-1) as:

$$I_{avg} = \frac{(0.5mA + 20mA) \times 30ms + (0.05mA + 0.01mA) \times 2300ms}{30ms + 2300ms} = 0.323mA$$

If 2.7Ah AA batteries were used to drive this demo board instead of the solar cells. This device could continue work about 8400 hours, approximately 348 days. Although the life time is extended compared with the 4 MHz CPU configurations, the total data volume is significantly reduced due to the longer cycle time mode necessary by repeated programming of the 60 bit RFIC register.

In the following part of this chapter, the test in the lab tests and outdoor environment tests will show the performance of 400kHz CPU clock EHR board.

2.5 Test Results

The measurements of this EHR demo board include Lab and outdoor performance tests. In the Lab test, we focused on the center frequency, duty cycle, board power consumption, and system reliability. The outdoor performance test was focused on the transmitting distance, and propagation issues inside and outside of the building.

2.5.1 Lab Test

2.5.1.1 Current Consumption Test

The Microcontroller and the RFIC are the two main power consumption components of this demo board. We already knew the RFIC's action currents was less than 20mA and the SLEEP mode currents was less than 0.01mA. The Microcontroller PIC power consumption was measured by using the photo board shown in Figure 2-12. This picture shows 4MHz clock configurations current consumption. The ACTIVE mode current was 0.74mA and SLEEP mode was 0.16mA (higher than expected from the datasheet). Use the same technique, we measured the 400kHz CPU clock configurations current consumption. The ACTIVE mode current was 0.5mA and the SLEEP mode was 0.05mA.

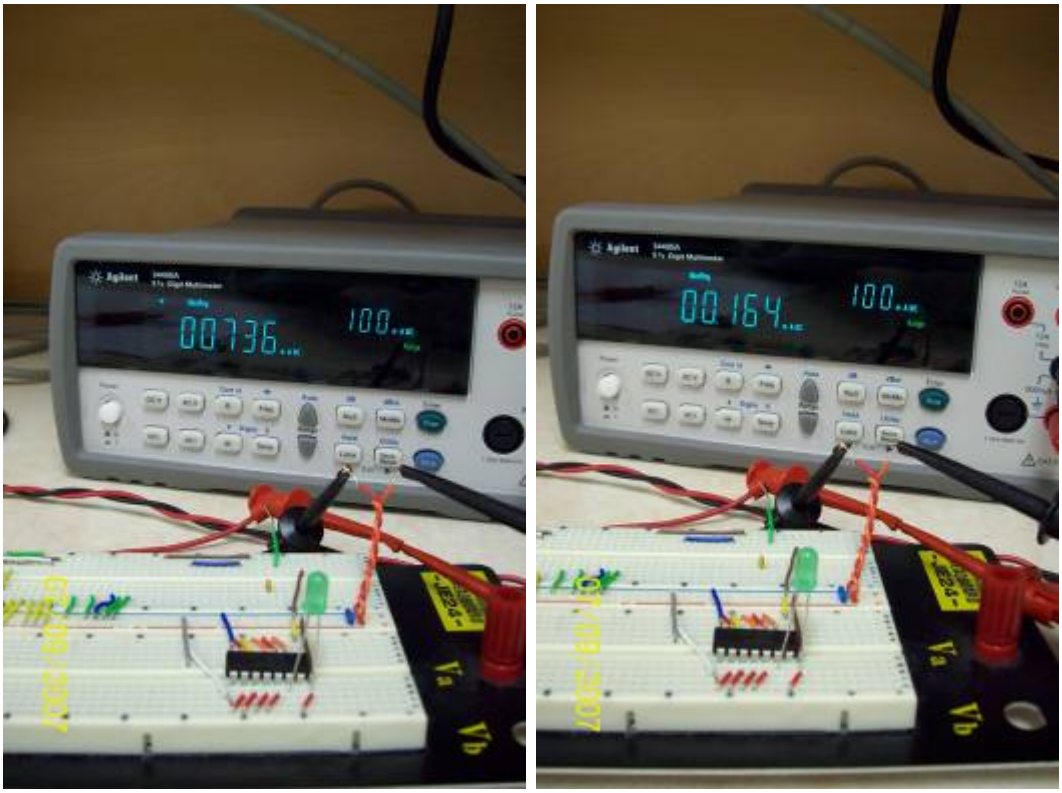


Figure 2-12 Microcontroller PIC16F676 Current consumption measurement with 4MHz clock (a) ACTIVE mode current (b) SLEEP mode current

2.5.1.2 The Burst Communication Test

The Burst communication was measured by using a Tektronix TDS 724D Oscilloscope, which is a two channel digital phosphor 500MHz~2GS/s oscilloscope. Figure 2-10 previously showed the duty cycle design of using a 400kHz CPU operating clock configuration.Figure 2-13 is 4 burst signal capture. The channel 1 signal is microcontroller control signal (Command signal). The channel 2 signal is the RF output burst signal.

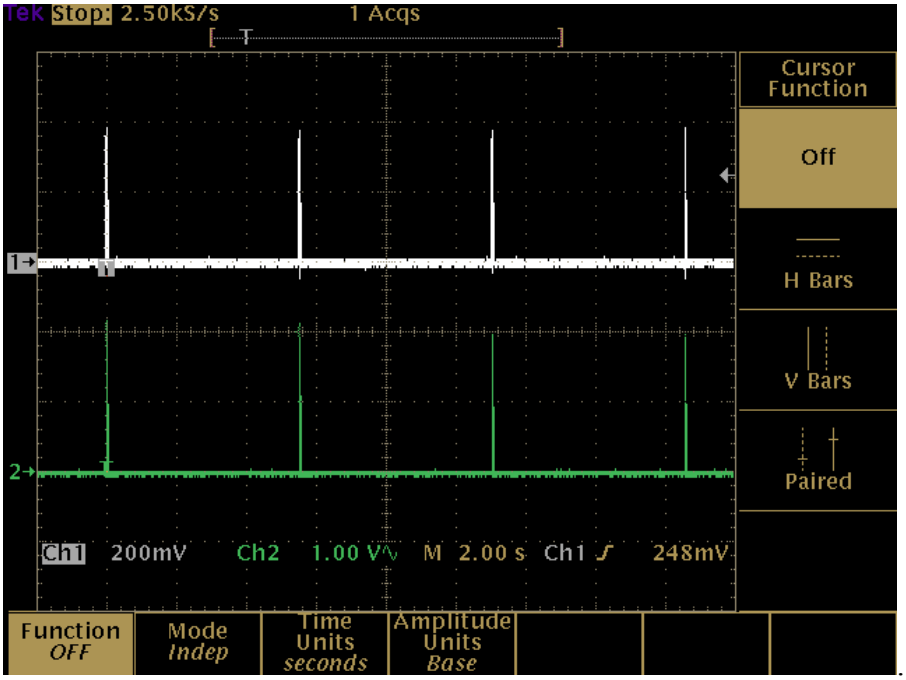


Figure 2-13 Four burst clusters with 1% duty cycle

Figure 2-14 is the zoomed in picture of one burst of the four in Figure 2-13. Channel 1 is the Microcontroller programming signal (Command signal) and the channel 2 is the RF output burst signal. Compared with Figure 2-10, it matches its design. The whole active period is 30ms.

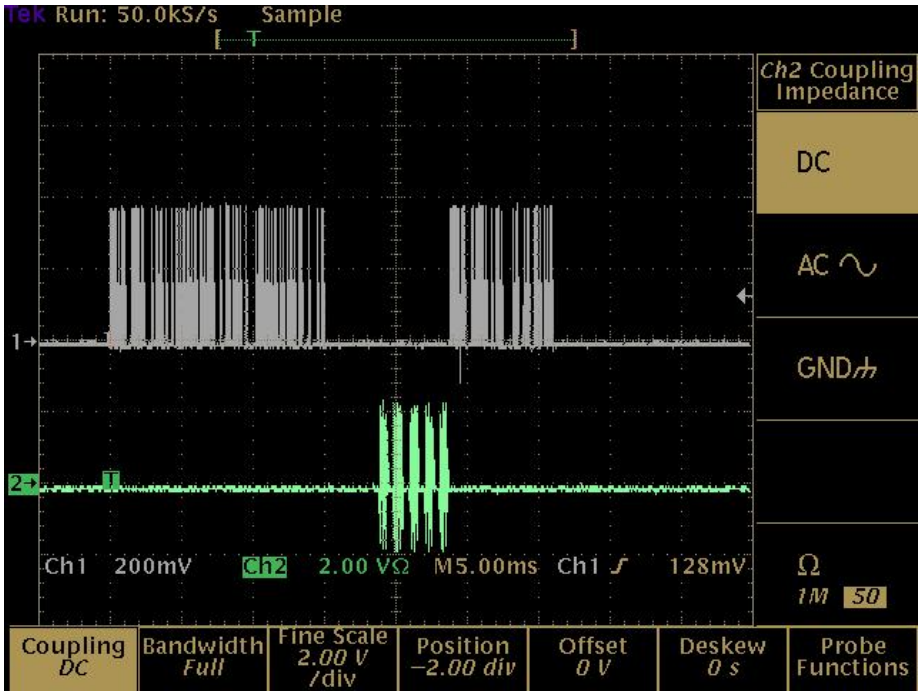


Figure 2-14 One Active period command signal and burst signal of 400kHz Clock Speed design

Figure 2-15 is the zoomed in burst period. Five bursts plus the time period of waiting TCXO standby, which last 8.68ms. The design time was 8ms. The difference was due to delay caused by software and oscilloscope measurement.

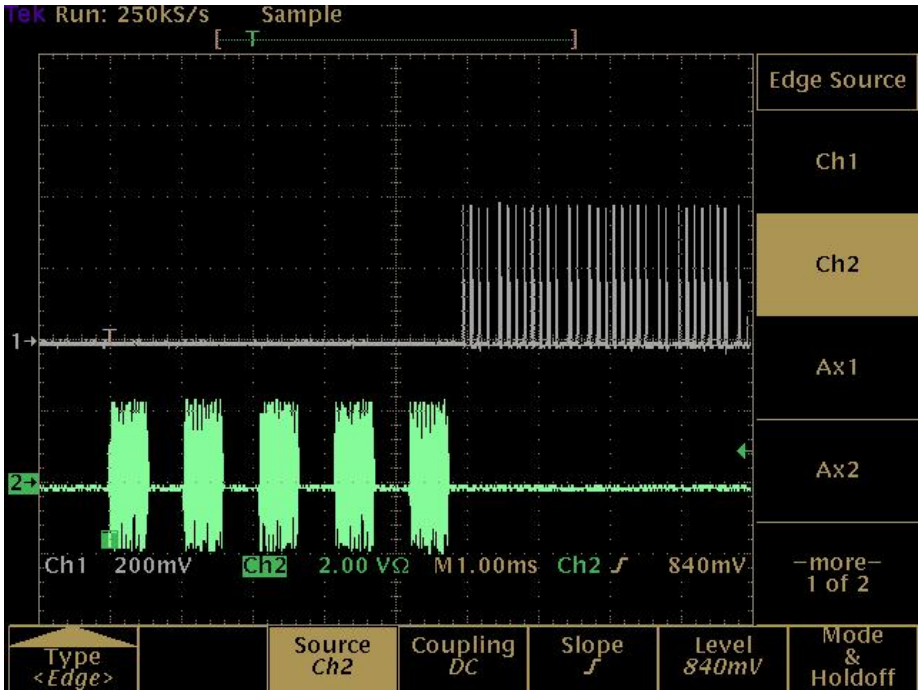


Figure 2-15 Five burst period

Figure 2-16 is the zoomed in picture of one burst signal. It is a 433MHz modulated sinwave.

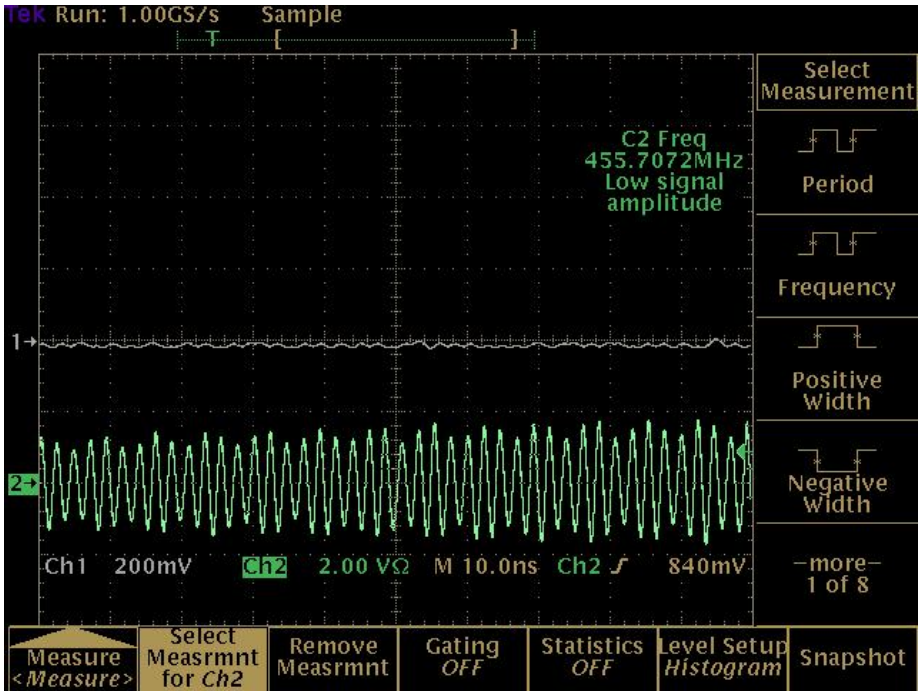


Figure 2-16 433.92MHz modulation signal

2.5.1.3 The System Reliability Test

For EHR systems, one important capability is that the system re-starts when energy satisfies the requirement after it was off. To test this, we moved the demo board away from the lights for about 5 minutes, thus the transmitting was stopped. When we move it back under to the lights, it successfully transmitted with expected duty cycle after the capacitor reached full charge.

2.5.2 Outdoor Test

The purpose of outdoor test was to estimate the transmitting range of the EHR prototype board. To test the range, we used a Yaesu VR-120D handheld radio shown in Figure 2-16. The VR-120D specifies a sensitivity of 0.6uV, which equates to -111dBm [11]. Since the burst clusters are actually are AM modulated signals we could hear a 'beep' when we use an AM receiver.

Figure 2-17 Yaesu VR-120 Receiver

Figure 2-18 are google maps with a mark to show the furthest point that the 'beep' could be heard. The signal was transmitted from within room RA2097 of Rathbone Hall at Kansas State University and received in the North parking lot up to approximately 0.2km away.

Figure 2-18 EH demo board transmitting range test (a) Satellite view of 0.2km transmission range (b) view look back from parking lot (c) view looking inside hallway of RA2097 [11]

2.6 Conclusions

This demo board has shown the feasibility of a simple energy harvesting wireless node with solar energy harvesting, which uses burst communication mode with very low duty cycle. This prototype board has shown the basic hardware skeleton of an energy harvesting wireless node including solar cell energy harvester, energy storage, radio and control unit. This chapter also has shown the software skeleton of an EHR node, which includes energy storage and management, burst communication and duty cycle management. The software allowed the system to self-recover after the power was re-satisfied. This demo board also proved that the K-State 400MHz RFIC is a very good RF front-end working in the energy harvesting applications as it has low active and sleep mode power, easy control to implement burst mode, and reliable performance.

CHAPTER 3 RF Link Budgets of EHR Systems

In the previous chapter, we demonstrated basic transmission performance by using ordinary ASK modulation techniques at 433.92MHz. However, the current IEEE802.15.4 standard defines 806MHz, 902MHz, and 2.4GHz as allocations for wireless sensor networks. Therefore, we investigated which frequency band is most applicable for EHR systems. To answer this question, we must understand the properties of electromagnetic waves and their propagation in an urban environment, in other words the RF Link budget in an urban environment must be understood. For RF communication, electromagnetic wave propagation considerations in the presence of natural and man-made structures heavily affects system performance, such as transmission range, and power consumption. Additionally, antenna technology will need to be considered.

There are many radio propagation models for digital wireless communication, such as Longley-Rice model for outdoor, Partition Losses (same floor) for indoor,etct, which provide a link model based on random variables and random processes. Such models allow people to simulate the RF signals on computers to improve the real world network deployments. But this is outside of our research. We are not designing a channel model here and will not discuss it. Instead, we take a measurements-based approach.

In this chapter, four unlicensed-band frequencies were picked in the VHF/UHF spectrum band, 151MHz, 433MHz, 902MHz and 2.4GHz and measured propagation data are reported and compared under two different RF link conditions. A custom 10mW signal source was used as the transmitter, and a spectrum analyzer as the receiver to provide accuracy to within 1dB and sensitivity to better than -120dBm. Both monopole and low-gain directional antennas were employed to represent expected use in an energy harvesting application and the measurement environment included both indoor and outdoor features.

3.1 RF Link Budget Factors

The choice of frequency spectrum is a fundamental factor for any radio communication link. Theoretically, an RF communication system can use any ISM (Industrial, Scientific and Medical) frequency for unlicensed applications such as EHR in wireless sensor networks.

However, picking a frequency for a radio system is not merely dependant on the frequency spectrum allocations available. There are several factors that we should consider, which include *frequency propagation environment versus transmitting range*, *antenna size, energy efficiency versus spectrum efficiency, and system complexity*. In this section, these factors are considered through an examination of *propagation and antenna technology* first*, then evaluating a RF link budget,* and finally considering a *radio system's complexity*.

3.1.1 Propagation

Transmitting range is always a key performance consideration of a wireless communication system. Electromagnetic wave propagation is one of the significant factors affecting transmitting range. An electromagnetic wave has a much more complex structure than many other waves (ex. sound waves) [13], and is described by Maxwell's equations. Maxwell's equations state that a changing magnetic field produces an electric field and a changing electric field produces a magnetic field, which indicates that electromagnetic waves are able to self-propagate [12]. However, the detailed interaction of these waves with the environment, radiation path, and antenna techniques used will all affect the transmitting range.

In free space, the electromagnetic waves are described as spherically propagating outward from the source in all directions. The instantaneous rate of energy flow across a area is ***P*** which is represented by Poynting's vector equation (3-1) where ***E*** is electric intensity and ***H*** is the

$$P = E \times H \tag{3-1}$$

magnetic intensity [13]. Ideally, the power density on the surface of an imaginary sphere can be expressed as equation (3-2) where ***S*** is the power density on the surface of the sphere in

$$S = \frac{P}{4\pi d^2} \tag{3-2}$$

watts/m^2, ***d*** is the diameter of sphere in meters, and ***P*** is the total transmitted power. Thus the Equ(3-2) shows the power density of the electromagnetic wave is inversely proportional to $\boldsymbol{d^2}$. But, in the real world, the electromagnetic waves are not radiating in an ideal free space environment, many different propagation issues occur, such as line-of-sight (LOS) propagation, obstructions, diffraction, refraction, tropospheric and ionospheric propagation. To analyze all those propagation models is beyond the scope of this research. We only consider the simple model of (3-2), and its varients where the propagation exponent is replaced by values up to 3 or 4

to capture such effects. The power at the receiver end may be expressed by equation (3-3), which indicates that the path loss equals the ratio of transmitted power to received power ratio.

$$L_P = \frac{P_T}{P_R} \tag{3-3}$$

The free space path loss (FSPL) is the signal strength loss when the electromagnetic wave radiates in free space. Equation (3-4) shows FSPL with frequency ***f***.

$$L_{FS} = (\frac{4\pi d}{\lambda})^2 = (\frac{4\pi df}{c})^2 \tag{3-4}$$

Here, ***d*** is distance between transmitter and receiver, ***λ*** is the signal's wave length, ***f*** is frequency of operation, and ***c*** is speed of light in vacuum (3.0E8 m/sec). This formula indicates the relationship between operation frequency and transmitting range. *From (3-4) we see that FSPL is proportional to the square of the operation frequency*. In other words, higher operation frequency leads to bigger free space loss, which leads to requirements for higher transmitting power and energy.

3.1.2 Antennas

Antennas are another important factor affecting transmitting range and link budget. It is the interface between air and physical radio systems. For every wireless communication system, antennas must be employed to radiate and receive electromagnetic energy. There are several fundamental concepts of antennas that should be explained so that a RF link budget can be evaluated.

3.1.2.1 Reciprocity

A fundamental principle of antennas, called *reciprocity*, states that antenna performance is the same whether radiation or reception is considered [14]. This principle states that the measurement of antenna parameters, such as gain, and beamwidth, are the same for both transmit and receive.

3.1.2.2 Antenna Directivity

In equation (3-2), the assumed radiation of electromagnetic wave has the same power density on all parts of the surface of a sphere. This would be called an isotropic radiator. The

corresponding antenna is called Isotropic Antenna. Real world antennas are not isotropic. They concentrate energy in certain directions. This effect is capture by the concept of antenna gain.

3.1.2.3 Gain

The antenna gain, usually called directivity gain, is defined as the ratio of the radiated power density at distance, ***d*** in the direction of maximum intensity, to the average power density over all angles at distance, ***d*** [12]. This is expressed by equation (3-5).

$$G = \eta \frac{P_{dirc}}{\frac{P_T}{4\pi d^2}} \tag{3-5}$$

Where $\boldsymbol{P_{dirc}}$ is the power density at d in maximum direction, $\boldsymbol{P_T}$ is the power applied to the antenna terminals, $\boldsymbol{\eta}$ is the total antenna efficiency which accounts for all losses in the antenna (which includes mismatch losses, conduction losses, and dielectric losses [15]). The denominator part is the average power density factor, which is calculated by using ideal isotropic antenna power density equation (3-2).

3.1.2.4 Effective Area

The effective area A_e or is introduced to determine the amount of power P_r that a receiver intercepts from a signal with power density P_{dirc} passing its location. Effective area can be defined by (3-6).

$$P_r = P_{dirc} A_e \tag{3-6}$$

where The relationship of effective area and antenna gain is expressed by equation (3-7) where ***G*** is antenna gain (not in decibel) and $\boldsymbol{\lambda}$ is wavelength.

$$A_e = \frac{\lambda^2}{4\pi} G \tag{3-7}$$

3.1.2.5 Antenna Size

The electrical length of a monopole antenna can be determined from basic antenna theory, which says the antenna's length λ/4 is inversely proportional to its operating frequency ***f*** according to equation (3-8) where c is the speed of light, $\boldsymbol{\lambda}$ is wavelength and ***f*** is the operation frequency.

$$\lambda = \frac{c}{f} \tag{3-8}$$

Obviously, the antenna size is inversely proportional to the operation frequency also.

3.1.3 Link Budget

We have explained and reviewed free space RF propagation and basic antenna techniques. It is time to move on into a link budget evaluation. The link planning is an essential part of a wireless communication network deployment, which will help to avoid resource waste, overdesign and poor system performance. A RF link budget is prepared in such a way that accounts for the transmitter radiated power and all of the losses in the link prior to the receiver [16]. It is not including any components of the noise figure, or digital link loss. A simple RF link budget can be expressed as equation (3-9),

$$P_{RX}(dBm) = P_{TX}(dBm) + G(dB) - L\ (dB) \tag{3-9}$$

where $\boldsymbol{P_{RX}}$ is the received power in dBm, $\boldsymbol{P_{TX}}$ is the transmitted power, $\boldsymbol{G}$ is product of TX and RX gains in the system (antenna gain, etc.), and $\boldsymbol{L}$ is loss (path loss, cable loss, etc.). In the RF link budget, the link margin reflects the robustness of a link, which is expressed by equation (3-10):

$$LinkM\arg in = EIRP - L_{Path} + G_{RX} - TH_{RX} \tag{3-10}$$

$\boldsymbol{L_{Path}}$ is free space path loss in dB, $\boldsymbol{G_{RX}}$ is the receive gain in dB, $\boldsymbol{TH_{RX}}$ is the receiver threshold or the minimum received signal level in dBm. The ***EIRP*** is the transmit power plus the transmitter antenna gain, minus any waveguide and random losses [12]. We do not consider any modulation and digitization effects in this RF link margin calculation.

3.1.3.1 Antenna Gain Effects on Link Budget

The free-space path loss is already known from equation (3-4). In RF link budgets, since antenna technology is involved, the Friis transmission equation can be modified to express the path loss with antenna gain in equation (3-11) where $\boldsymbol{G_T}$ is transmitter antenna gain, $\boldsymbol{G_R}$ is receiver antenna gain (not in dB).

$$L = \frac{P_T}{P_R} = \frac{1}{G_T G_R}\left(\frac{4\pi d}{\lambda}\right)^2 \tag{3-11}$$

Thus, considering the discussion above about antenna effective area, combining equation (3-6), (3-7) and (3-11), we could rewrite the free space loss as equation (3-12)

$$L = \frac{P_T}{P_R} = \frac{(4\pi d)^2}{G_R G_T \lambda^2} = \frac{(4\pi)^2 d^2 f^2}{G_R G_T c^2} \tag{3-12}$$

3.1.3.2 Link Budget Examples

From the analysis above, we can obtain the following results:

First, for a set distance between the transmitter and receiver, since the gains and effective areas are constant, *the path loss is inversely proportional to the operation frequency **f***.

Second, from previous antenna parts, we know that *the antenna size will increase with lower operation frequencies*. For example, a half-wave dipole antenna at 3GHz is a manageable 5cm in length, whereas at 300MHz it is 50cm in length. Fortunately, the antenna physical size could be shrunk by using special material [17] or some technique like non-planar rings [18].

Third, is equation (3-12), $\mathbf{d^N}$ where **N=2**, **N** is called the path loss exponent, which *is for RF propagation in free space*. However, in the real world, there are lots of reflections, diffractions or multipath, and this exponent will vary *from 1 to 6* depending on different environments. By using equation (3-11), we can easy calculate the transmitted power versus transmitting range. Assume $\boldsymbol{P_R=10P_n}$ while $\boldsymbol{P_n=kTB}$ and the ideal antenna gain $\boldsymbol{G_T}$ and $\boldsymbol{G_R}$ are 1, k is 1.38E-23 and ***T*** is 290k, ***B*** is 1kHz. Figure 3-1 and 3-2 are 433MHz and 2.4GHz plots of transmitting range versus transmit power with 2 to 4 path loss exponents.

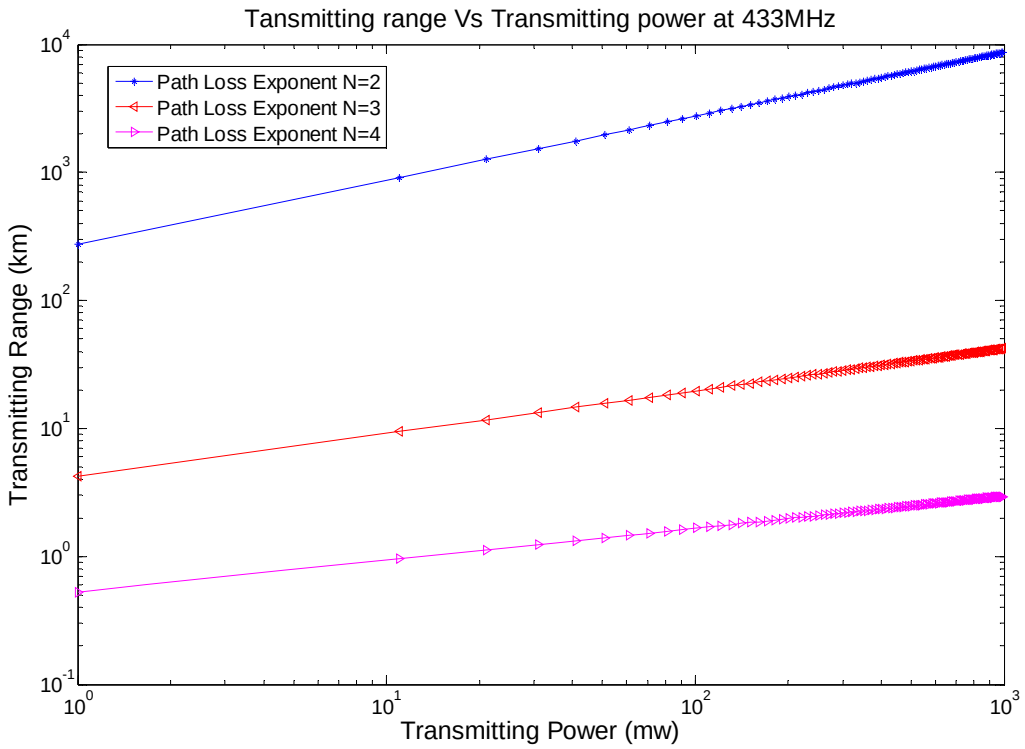


Figure 3-1 Calculated range at 433 MHz with dipole antennas and 1 kbps data rate

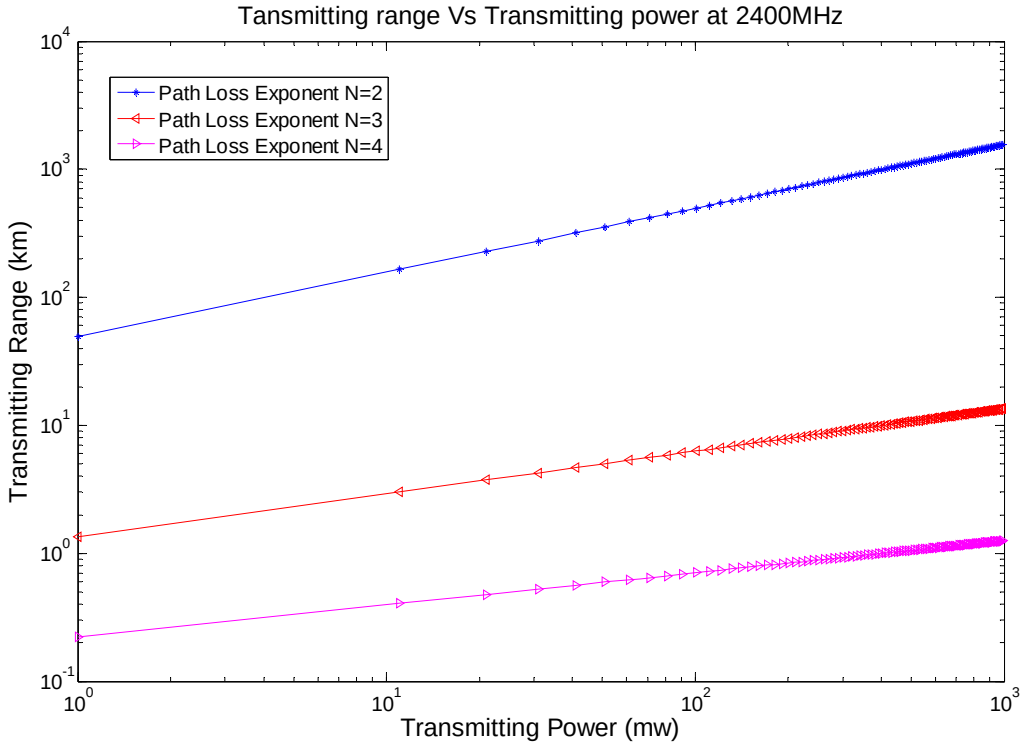


Figure 3-2 Calculated range at 2.4 GHz with dipole antennas and 1 kbps data rate

3.1.4 RF Link Budget and The System Design of EHR

So far, we have shown that lower frequency can transmit further than higher frequency under same power level, when at least one of the antennas used is low-gain. Unfortunately, most commercial WSN products are using the crowed high frequency bands of 800-900MHz, 902-928MHz (Europe), or 2.4-2.5GHz. The main reasons are because these bands are world wide allocation where inexpensive radio transceivers are commercially available and higher data rate can be provided [19]. But the negative aspect of using these frequencies is that the range is fundamentally limited and interference is very serious from higher power transmitting systems, such as WLAN and other fixed transmitters. This is potentially harmful for the power-constrained EHR systems since the lower power sensor nodes are easily affected by stronger signals.

Obviously, there are many techniques which can be used to detect and recover signals within low SNR environment in such crowed frequency bands, such as using MIMO (multiple input and multiple output) antenna systems, using CDMA, FDMA or OFDMA multi-access techniques, or using convolutional codec etc. But, at the same time, these techniques are increasing the complexity, and therefore power consumption of system design. Typically, the complex PHY and MAC layer protocol standards are defined assuming battery powered operation. On the hardware side, to implement these standards, a wide-band RF front-end, a high clock rate FPGA and DSP baseband processor are required. On the software side, a real-time operating system, complex codec algorithms and multi-tasking protocol stacks must be implemented that need be run at higher CPU clock rates to guarantee the wireless communication real-time capabilities.

Thus, EHR system designs must consider the system complexity. We propose to use low data rates along with low RF operation frequency because it leads simple, low complexity implementations, which includes lower power hardware and lower MIPS DSP software design. In the following subsections, four unlicensed VHF/UHF frequencies propagation are measured and compared by using practical equipments and methods. It will give a practical proof of lower frequency's benefit for energy constrained radio systems. In the subsequent chapter, the issues of low-complexity PHY and MAC layer techniques are considered.

3.2 UHF/VHF Propagation Comparisons

For this research, we selected frequencies of 151MHz, 433MHz, 902MHz and 2400MHz for several reasons: First, in 2009 significant new spectrum resources will be freed-up in the UHF frequency range in the United States. Around the world, similar and even more dramatic changes are occurring [20]. Second, the 2400MHz and 902MHz are ISM frequency band allocated to IEEE802.15.4 standard, which is widely used standard of wireless sensor networks. 433.92MHz is in an unlicensed band widely used for remote-sensing and RFID applications. In light of these issues and the previous analysis of the relationship between frequency and energy harvesting radio system, we have undertaken a fresh look at which frequencies are most appropriate for energy constrained radio systems.

3.2.1 Experimental Setup

3.2.1.1 Propagation Links

Two different links were measured and compared as shown in Figure 3-3. One link was measured by using monopole antennas at both transmitter and receiver end. The second link was measured by using monopole antenna at receiver end and directional antenna at transmitter end.

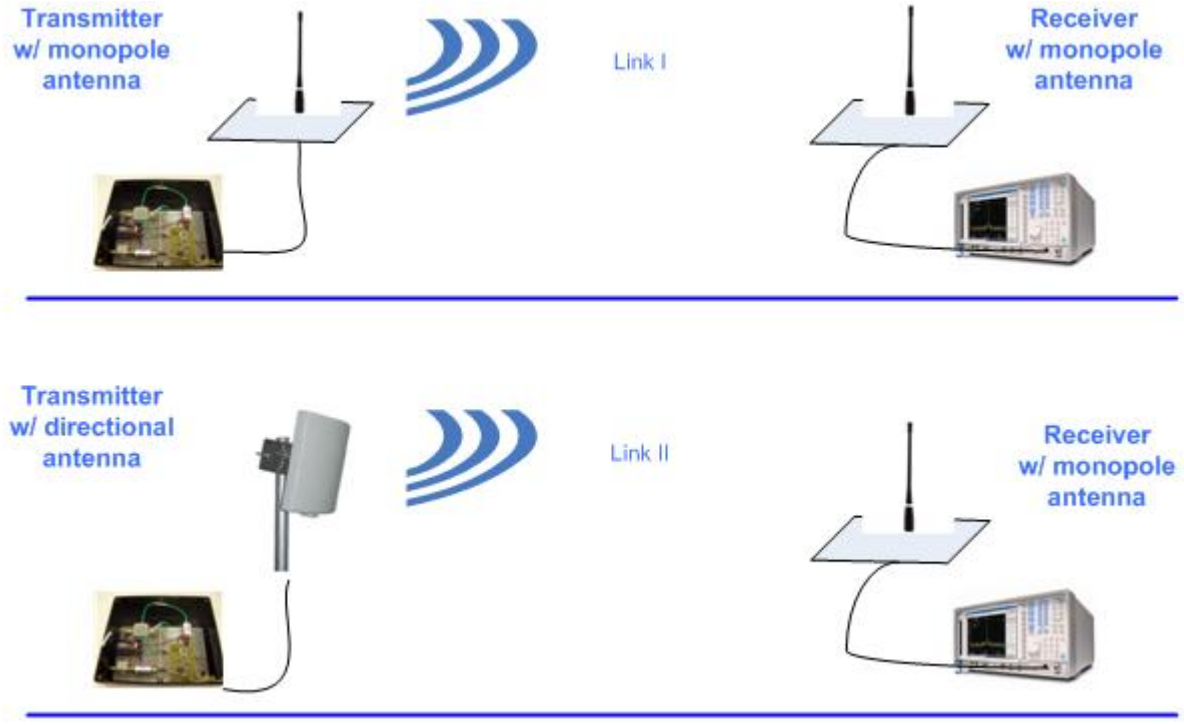


Figure 3-3 Two type of experimental Propagation Links

3.2.1.2 Experiment Environment

The testing locations were based at Kansas State University and included both indoor and outdoor propagation features. The receiving station was sited indoors, in room RA2097 (2,5 of Figure 3-4), on the 2nd floor of the engineering building This building contains four floors and consists of a concrete foundation, concrete wall supports, concrete slab floors with metal supports (3, 6, 7 of Figure 3-4) and dry wall partitions (4 of Figure 3-4). The spaces between floors and walls contain power and cable lines, air conditioning ducts, fire protection sprinklers, and steel piping (3,6,7 of Figure3-4). Due to its structural makeup, this building serves as a good representative testing environment for construction monitoring applications.

Figure 3-4 Rathbone Hall Engineering Building constructions

For indoor and outdoor measurements, several points at varying transmission radii away from the receiver were selected. Figure 3-5 shows the receiver measurement location mark on the Rathbone Hall 2nd floor plan and the outdoor map.

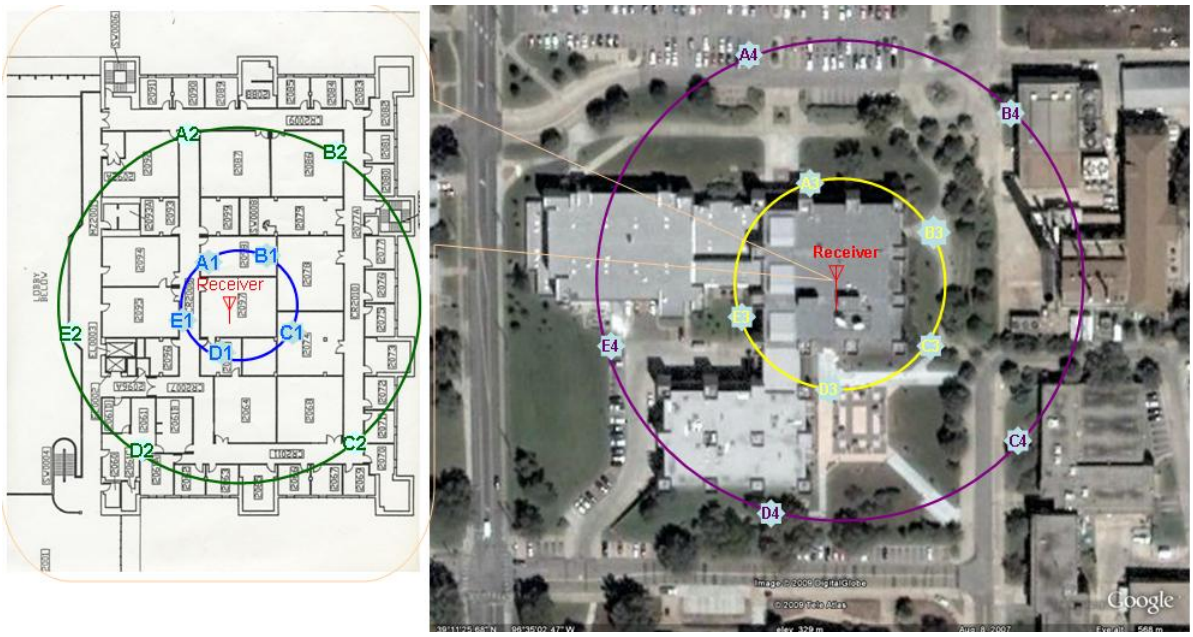


Figure 3-5 Measurements location marked on the indoor floor plan and outdoor map : Left is 2nd floor plan of Rathbone Hall; Right is the Google earth map of Rathbone Hall

The points A to E, five group points, were marked on the map. Each group has 4 points kept in line, to represent a propagation path, for example A1, A2, A3, A4, and A5.

Figure 3-6 Two path used for lower frequency transmitting range measurement

The outdoor measurement paths were located north and northeast of the Ratbone Hall (Figure3-6) due to the fairly open terrain, but included several obstructions such as buildings and trees. Two similar sized buildings are located north at distances of 0.16km and 0.27km while a taller building is located 0.21km north east.

The area is relatively flat with maximum elevation variations of approximately 10 meters. The path B is higher elevation than path B. Located east of the engineering building is the main campus whose buildings are far more congested.

3.2.1.3 Antennas

Both directional and non-directional antennas are used for this measurement. For the omni-directional tests, monopole antennas were constructed and used at both ends of the link. The antennas used as directional antennas were commercial panels and Yagis. Figure 3-7 shows antennas pictures used in measurements.

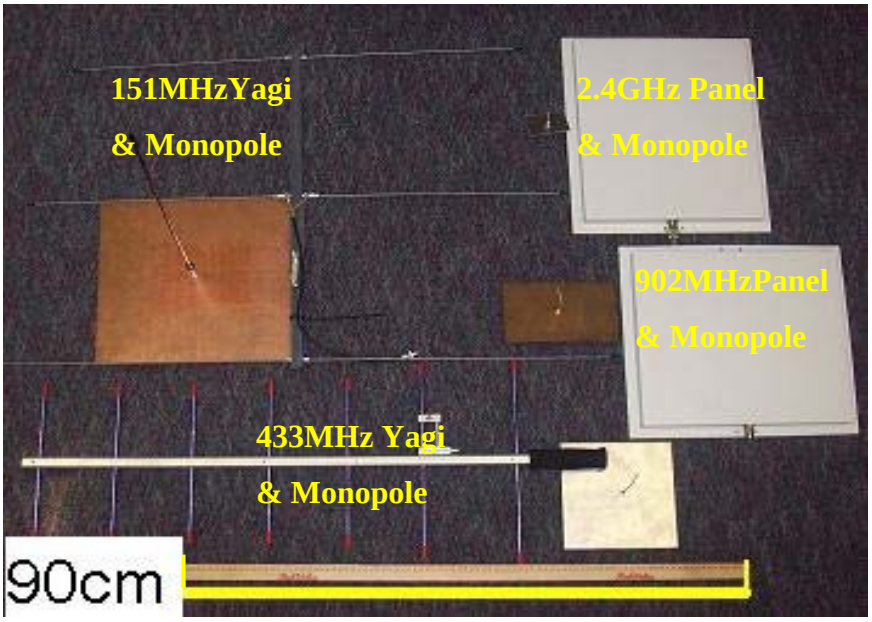


Figure 3-7 Antennas used in measurement

Table 3-1 shows the relationship between the measured frequency, quarter-wave length and the effective aperture of the antennas. For a half-wave length dipole antenna, the effective aperture is calculated by equation (3-6) when $G = 1.6$ [15].

Table 3-1 quarter-wave antenna size and effective aperture of dipole antenna

Frequency (MHz)	Quarter-wave Length λ/4 (cm)	Effective Aperture of Dipole Antenna (cm^2)

151.94	49.36	3102
433.92	17.28	380
902	8.31	97.92
2400	3.12	12.39

For monopole antenna, truncated ground planes were used. To validate the antenna constructions, reflection coefficients of all antennas were measured prior to use. Figure 3-8 shows the monopole antenna and the ground planes.

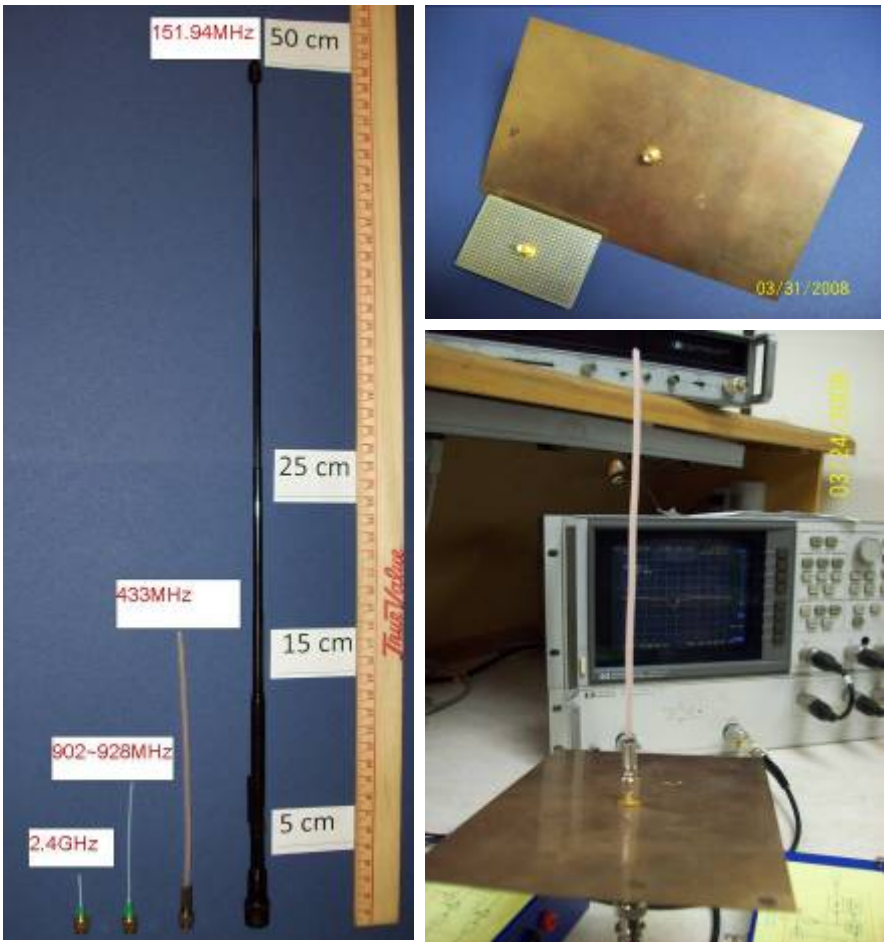


Figure 3-8 Monopole antenna, Ground plane and antenna reflection coefficients measurements

Figures 3-9, 3-10, 3-11 and 3-12 show 151MHz, 433MHz, 902MHz and 2.4GHz directional and monopole antenna reflection coefficient S11 measurement results. The S11 ranged from -15dB to -35dB.

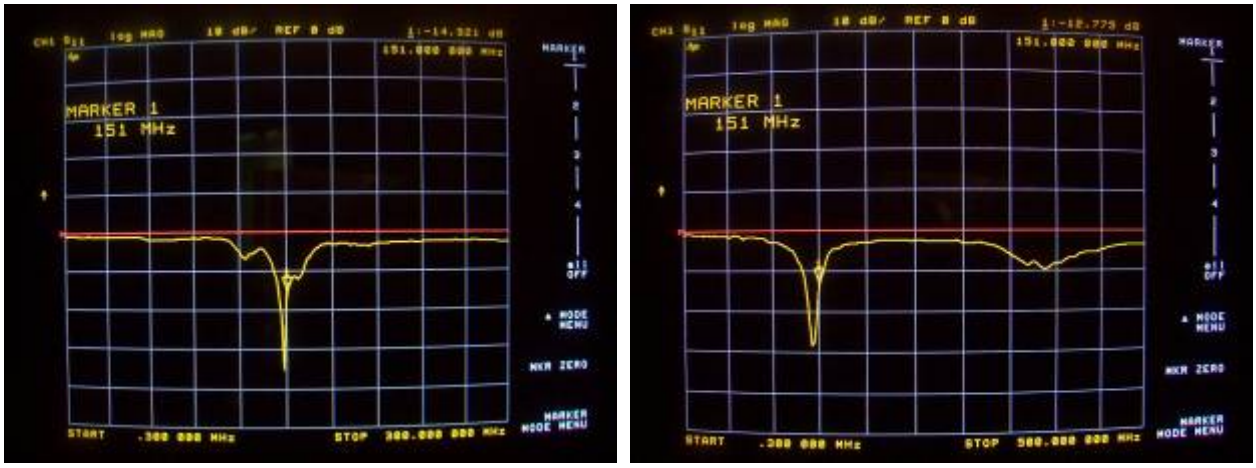


Figure 3-9 151MHz directional antenna S11=-14.9dB, monopole antenna S11=-12.7dB

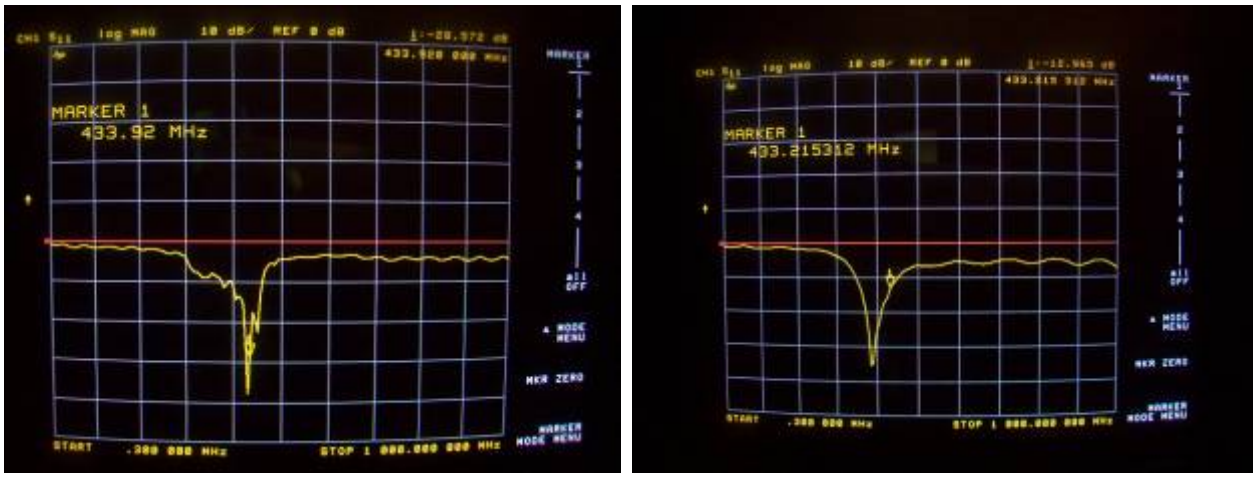


Figure 3-10 433MHz directional antenna S11=-28dB, monopole antenna S11=-12.9dB

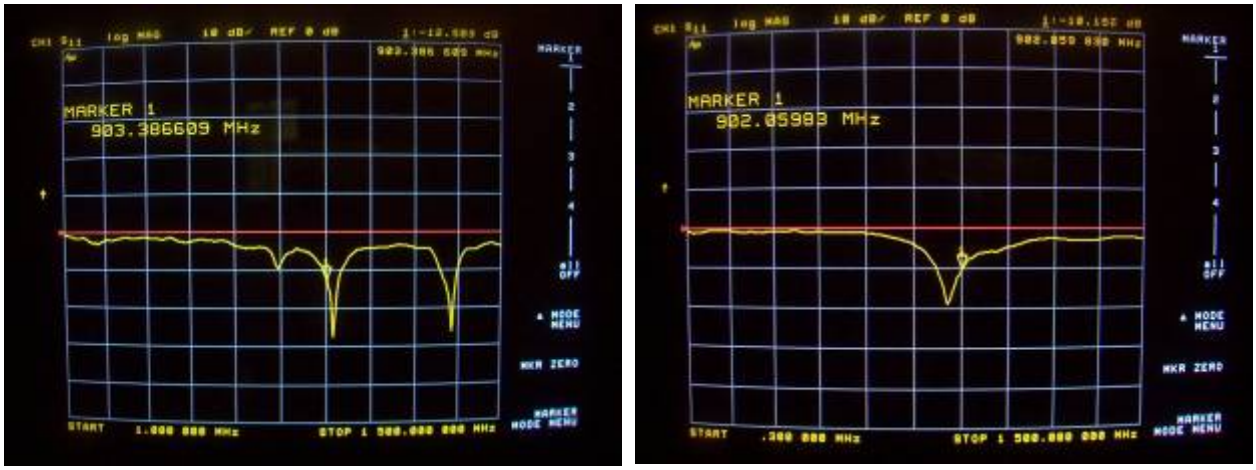


Figure 3-11 902MHz directional antenna S11=-12.5dB, monopole antenna S11=-10dB

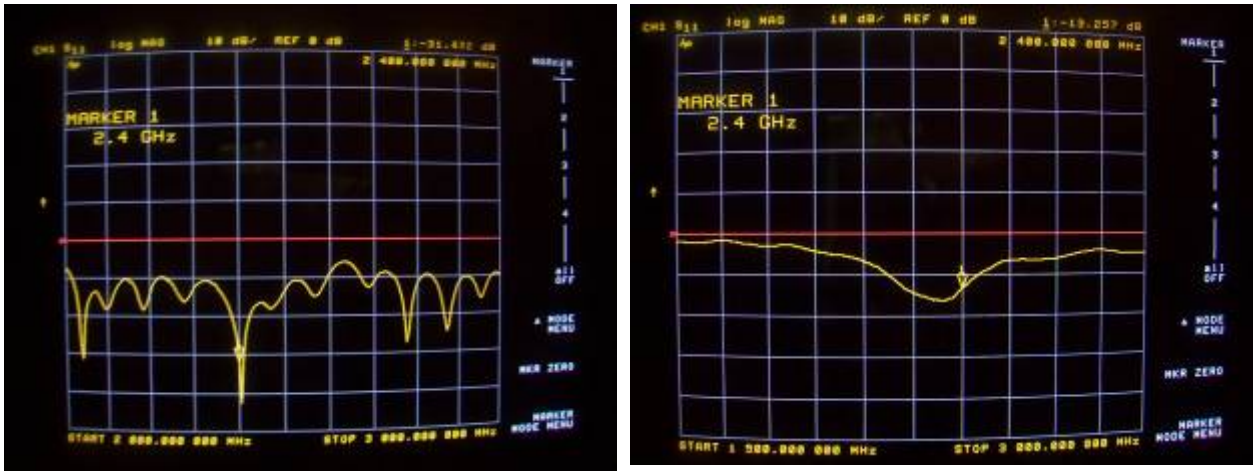


Figure 3-12 2400MHz directional antenna S11=-31dB, monopole antenna S11=-13.3dB

3.2.1.4 Transmitter and Receiver

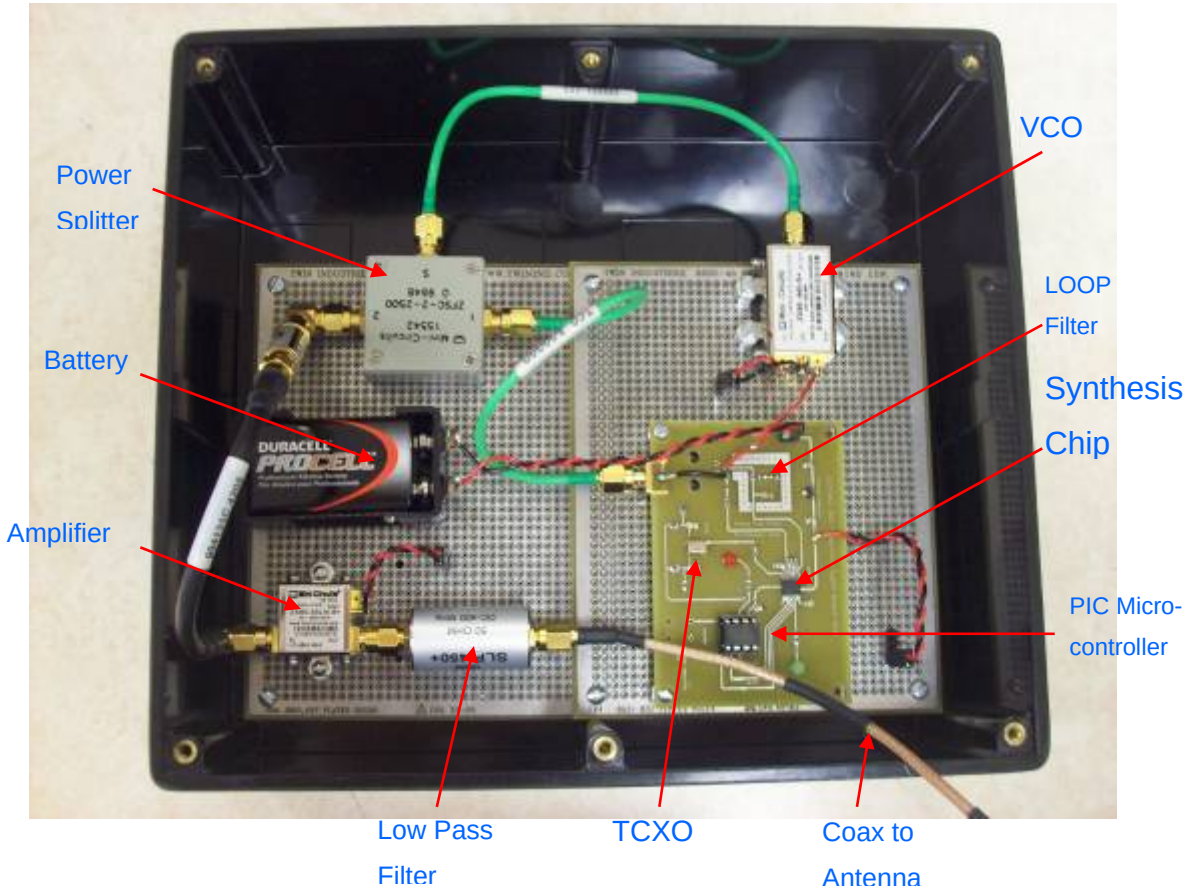


Figure 3-13 Portable 10mW Transmitter photo

The transmitters were designed to output 10mW un-modulated signals and were placed in boxes to make them more potable. The transmitter circuit includes a power supply, frequency

synthesizer board, amplifier, and an output harmonic filter. Figure 3-13 is the transmitter photo. They shared an LMX 2326-based frequency synthesis board, Mini- circuits 15542 power splitter, Mini-Circuit ZX60-33LN-S+ power amplifier, and PIC 12F509 microcontroller hardware. The VCO, Low Pass Filter and Microcontroller software are selected individually for each frequency. **Appendix C** shows the schematic and layout of frequency synthesis board. **Appendix F** provides the software code for the different frequency synthesizer boards. Table3-2 is the components list used to build four frequencies transmitter.

Table 3-2 Components list of 10mW un-modulated transmitter

<table>
<tr><th></th><th>VCO</th><th>LPF</th><th>AMP</th><th>PW Splitter</th><th>Synthesis Chip</th><th>TCXO</th></tr>
<tr><td>151MHz</td><td>ZX95-148+</td><td>SLP-200+</td><td rowspan="4">ZX60-33LN-S+</td><td rowspan="4">ZFSC-2-1W-S+</td><td rowspan="4">LMX2326</td><td rowspan="4">19.2MHz crystal oscillator</td></tr>
<tr><td>433MHz</td><td>ZX95-445+</td><td>SLP-450+</td></tr>
<tr><td>902MHz</td><td>ZX95-930C+</td><td>SLP-1000+</td></tr>
<tr><td>2.4GHz</td><td>ZX95-2550+</td><td>SLP-2950+</td></tr>
</table>

The receiver was an Agilent N9320 spectrum analyzer. The minimum receiver power of this equipment, with the preamp enabled, is lower than -130dBm so that it is adequate. Figure 3-14 is a photo of the receiver with a 433MHz monopole antenna attached. The other frequencies have similar receiver architecture except antenna length.

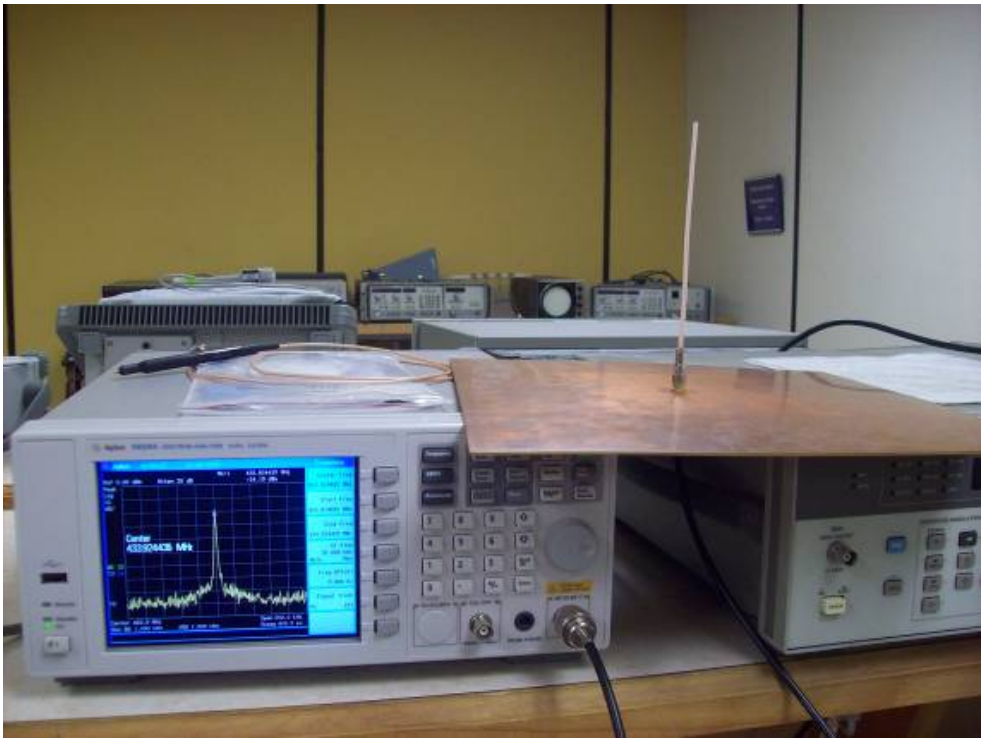


Figure 3-14 Receiver—Spectrum and monopole antenna

3.2.2 Experiment Results

The experiment covered indoor and outdoor environments up to a range of 1.5km. Using room RA2097 as the center location, four circles were scribed with two indoor (7m and 30m) and two outdoor (40m and 90m) radii. Each circle contains five measurement locations (Figure 3-5). Beyond 90 meters, the measurement locations were not fixed.

3.2.2.1 Received Signal Strength Measurements at Each Frequency

The propagation measurements for the four frequencies are recorded in Table 3-3, Table 3-4, Table 3-5 and Table 3-6 with both monopole and directional antennas. In the tables, the item Environment indicates indoor or outdoor depending on the location of the transmitter. Item Mark is used to mark the location on the map (Figure 3-5). Item Distance in meters is the range between transmitter and receiver. Item Power(dBm)_Drc is the received power when the directional antenna is used at the transmitter end. Item Power(dBm)_Mon is the received power when the monopole antenna is used at the transmitter end.

Table 3-3 151MHz Propagation measurement results

151MHz				
Environment	Mark	Distance(m)	Power(dBm)_Drc	Power(dBm)_Mon

Indoor	C2	7.74	-14	-13
	D2	7.74	-13	-20
	E2	7.74	-21	-23
	A2	7.74	-28	-24
	B2	7.74	-17	-21
	C	31.3	-30	-38
	D	31.3	-36	-42
	E	31.3	-31	-36
	A	31.3	-38	-44
	B	31.3	-32	-40
Outdoor	A3	43.86	-63	-74
	B3	46.36	-64	-68
	C3	52.65	-60	-74
	D3	41.36	-45	-59
	E3	45	-69	-61
	A4	85	-70	-74
	B4	84	-79	-82
	C4	83	-72	-86
	D4	90	-85	-92
	E4	83	-64	-82
	A5	129	-86	N/A
	A6	165	-86	N/A
	A7	323	-88	N/A
	A9	1100	-104	N/A
	A10	1130	-103	N/A
	A8	1460	-110	N/A

Table 3-4 433MHz Propagation measurement results

433MHz				
Environment	Mark	Distance(m)	Power(dBm)_Drc	Power(dBm)_Mon
Indoor	C2	7.74	-19	-27
	D2	7.74	-22	-25
	E2	7.74	-29	-27
	A2	7.74	-22	-33
	B2	7.74	-25	-29
	C	31.3	-36	-42
	D	31.3	-43	-50
	E	31.3	-39	-42
	A	31.3	-45	-54
	B	31.3	-43	-43
Outdoor	A3	43.86	-64	-75
	B3	46.36	-65	-74
	C3	52.65	-65	-77
	D3	41.36	-52	-65
	E3	45	-73	-84
	A4	85	-68	-80
	B4	84	-70	-80

	C4	83	-68	-77
	D4	90	-86	-96
	E4	83	-74	-85
	A5	129	-74	-83
	A6	165	-80	-89
	A7	323	-97	-95
	A433-8	496	-115	-120
	A433-9	672	-120	-122

Table 3-5 902MHz Propagation measurement results

900MHz				
Environment	Mark	Distance(m)	Power(dBm)_Drc	Power(dBm)_Mon
Indoor	C2	7.74	-21	-31
	D2	7.74	-31	-35
	E2	7.74	-29	-34
	A2	7.74	-32	-34
	B2	7.74	-31	-36
	C	31.3	-43	-49
	D	31.3	-55	-58
	E	31.3	-41	-51
	A	31.3	-53	-58
	B	31.3	-49	-57
Outdoor	A3	43.86	-72	-85
	B3	46.36	-63	-75
	C3	52.65	-71	-80
	D3	41.36	-61	-75
	E3	45	-73	-74
	A4	85	-78	-88
	B4	84	-77	-90
	C4	83	-72	-84
	D4	90	-95	-97
	E4	83	-71	-86
	A5	129	-86	-99
	A6	165	-95	-102
	A7	323	-109	-110
	A900-8	353	-114	N/A
	A900-9	379	-110	N/A
	A900-A	537	-120	N/A

Table 3-6 2400MHz Propagation measurement results

2400MHz				
Environment	Mark	Distance(m)	Power(dBm)_Drc	Power(dBm)_Mon

Indoor	C2	7.74	-32	-43
	D2	7.74	-45	-47
	E2	7.74	-41	-42
	A2	7.74	-39	-43
	B2	7.74	-44	-45
	C	31.3	-59	-69
	D	31.3	-59	-69
	E	31.3	-61	-69
	A	31.3	-65	-72
	B	31.3	-53	-63
Outdoor	A3	43.86	-92	-97
	B3	46.36	-80	-91
	C3	52.65	-91	-98
	D3	41.36	-75	-86
	E3	45	-78	-89
	A4	85	-82	-98
	B4	84	-92	-98
	C4	83	-93	-98
	D4	90	-98	-104
	E4	83	-90	-100
	A5	129	-82	-101
	A6	165	-88	-115
	A2400-7	198	-120	-125

3.2.2.2 Directional Antennas versus Monopole Antenna Results

Figure 3-15 shows 151MHz propagation comparisons using the directional versus non-directional antenna. The blue circles are for the directional antenna at the transmitter end.

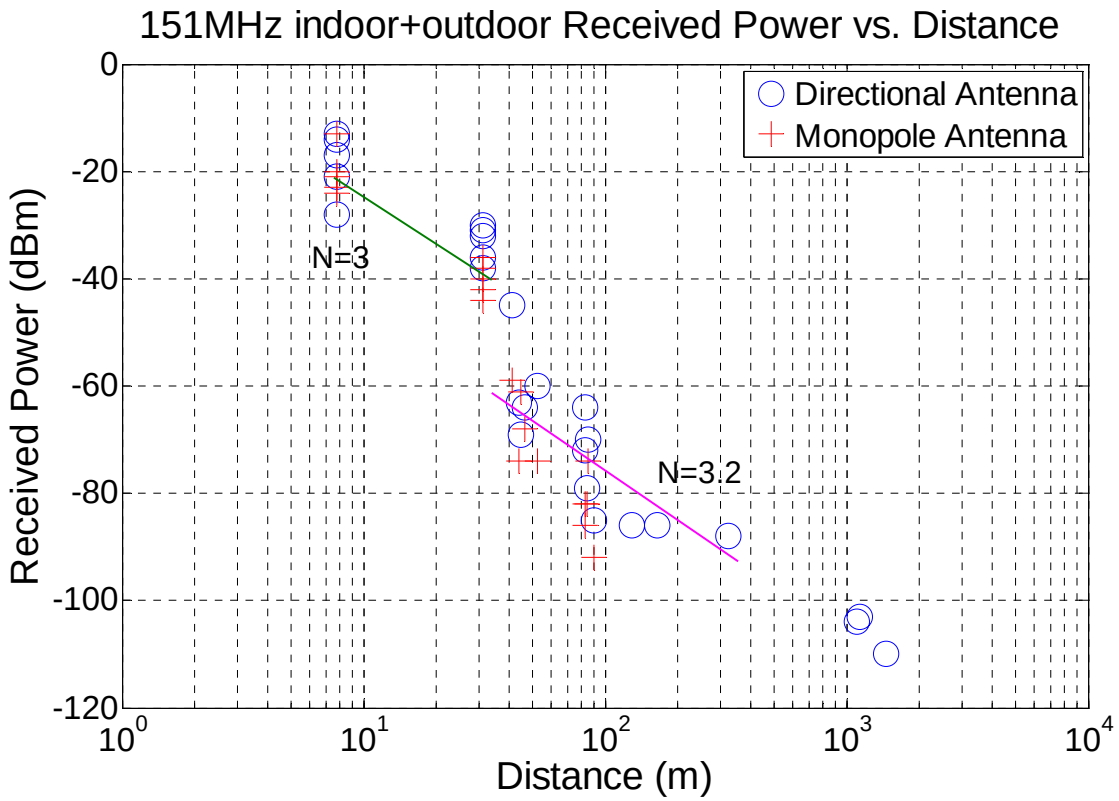


Figure 3-15 151MHz two different link propagation with Path loss Exponent

Obviously, using directional antenna can improve reception by almost 6dB over using the monopole antenna on average, which is consistent with the estimated gain of the Yagi beam antenna. The indoor path loss exponent N is 3 and the outdoor path loss exponent N is 3.2. Note that the received signal power experienced a 12 dB step decrease around 50m because of an indoor/outdoor boundary.

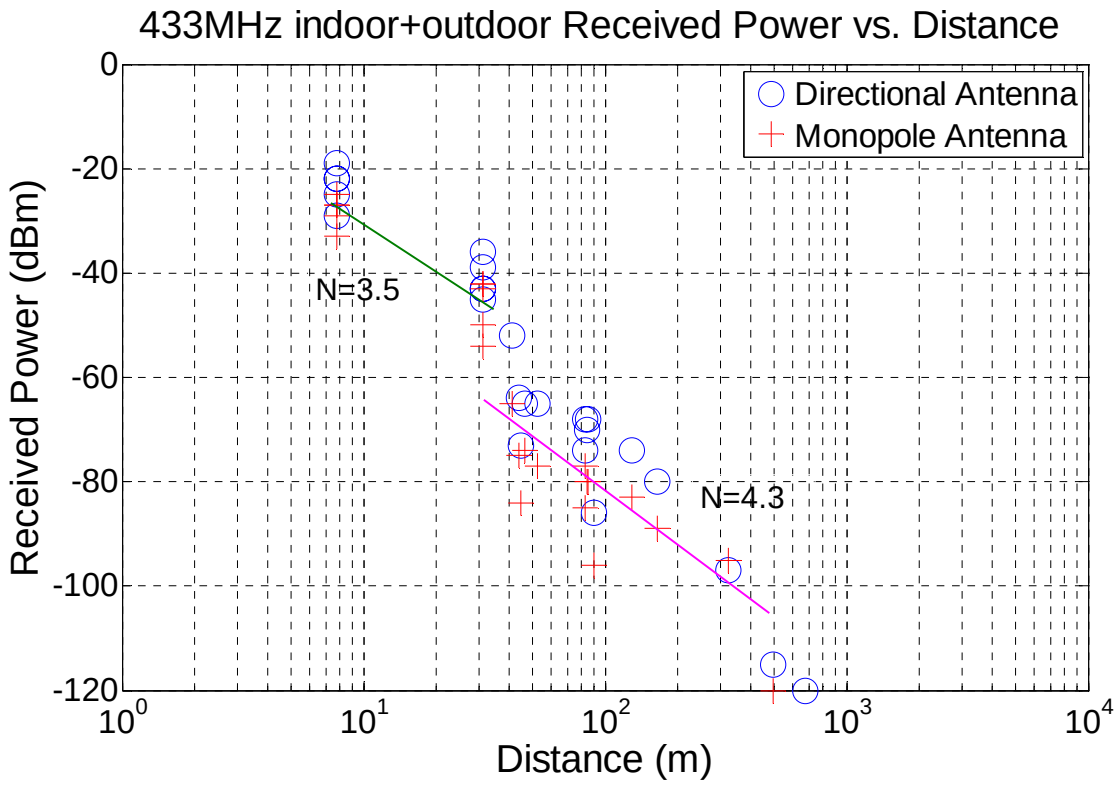


Figure 3-16 433MHz two different link propagation with Path loss Exponent

Figure3-16 shows results at 433 MHz. This data shows that using directional antenna can improve reception by almost 8dB over using the monopole antenna on average, which is again consistent with the estimated yagi antenna gain. The indoor path loss exponent N is 3.5 and the outdoor path loss exponent N is 4.3. The indoor/outdoor boundary produced a 10dB step of excess path loss.

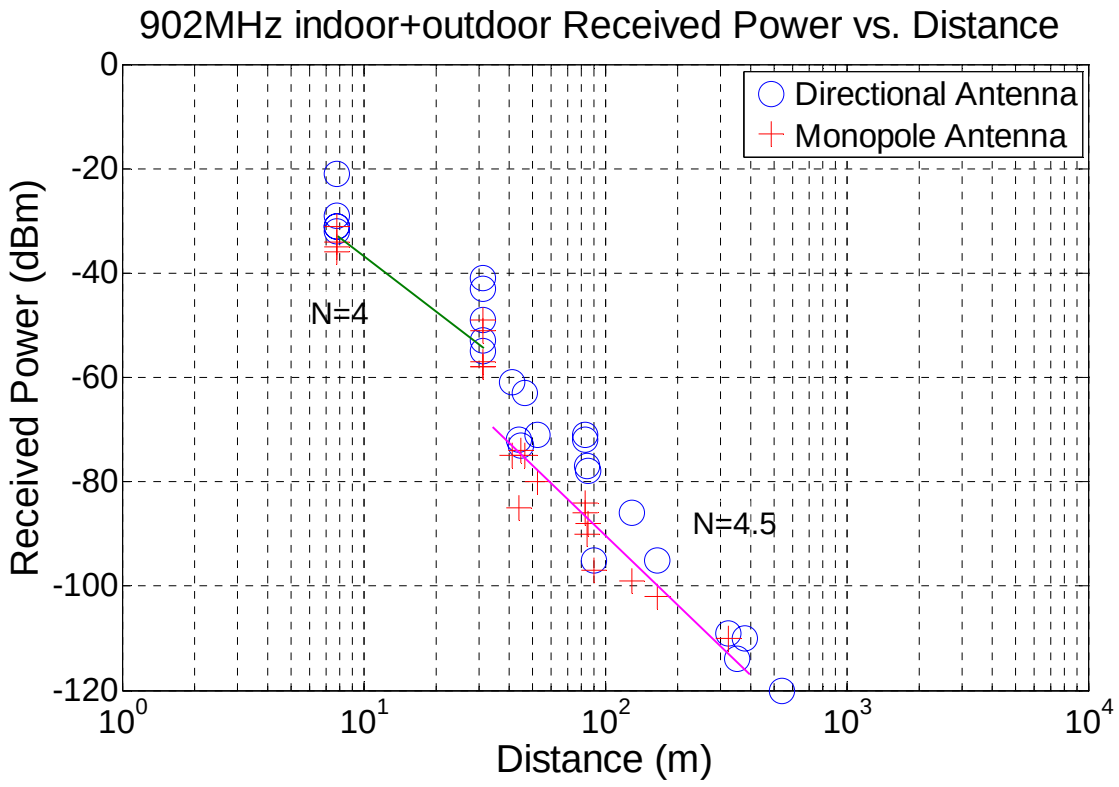


Figure 3-17 902MHz two different link propagation with Path loss Exponent

Figure3-17 shows that using flat-panel directional antenna (comparable in physical size to the yagis) can improve reception by almost 12dB over using the monopole antenna on average at 902 MHz. The indoor path loss exponent N is 4 and the outdoor path loss exponent N is 4.5. The indoor/outdoor boundary of 902MHz experienced a 20dB step of excess path loss.

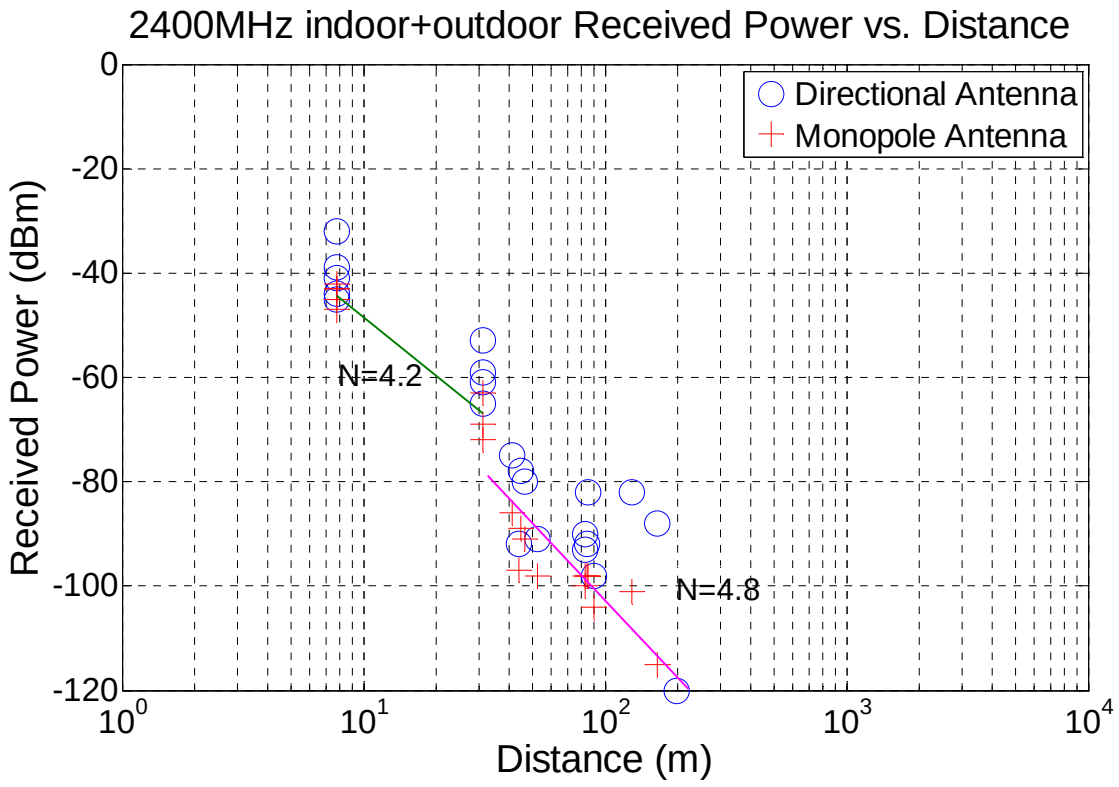


Figure 3-18 2400MHz two different link propagation with Path loss Exponent

Figure3-18 shows that using similar-sized directional antennas can improve reception by almost 12.5dB over using the monopole antenna on average at 2400 MHz. The indoor path loss exponent N is 4.2 and the outdoor path loss exponent N is 4.8. The indoor/outdoor boundary of 2400MHz experienced a 29dB step of excess path loss.

3.2.2.3 Path Loss Exponent Comparison

Table 3-7 shows path loss exponent comparison of all these four frequencies. From this table, we note that in addition to its theoretical free-space path loss disadvantages, the higher frequency suffers from higher path loss exponents in a terrestrial link environment. Thus, the advantages of lower frequencies are even higher than expected when both ends of the link use relatively non-directional antennas.

Table 3-7 Path Loss Exponents comparison

	151MHz	*433MHz*	*902MHz*	*2400MHz*
Indoor	3	3.5	4	4.2
Outdoor	3.2	4.3	4.5	4.8

3.2.2.4 Comparison of Four Frequency's Propagation

To more easily compare the four frequency's propagation, Figure 3-19 shows the results when the directional antenna is used at transmitter end and

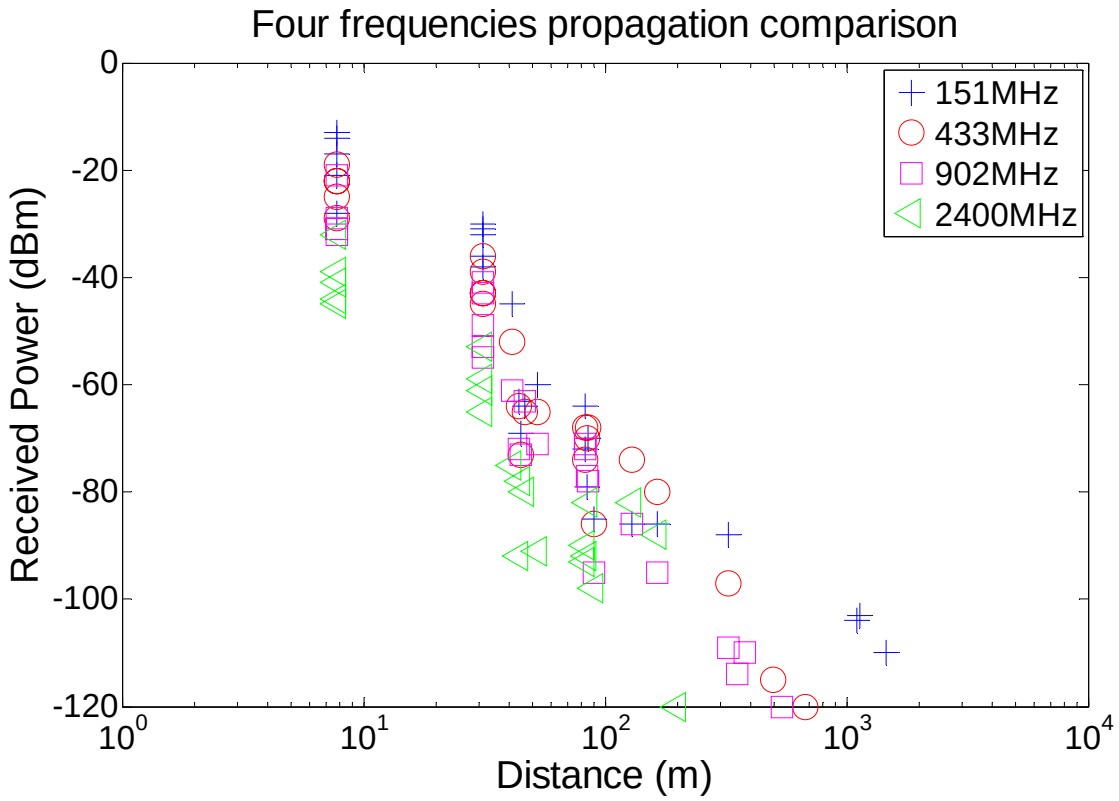


Figure 3-19 Four frequencies propagation comparison

monopole antenna is used at receiver end. From the Figure 3-19, on average, the 151MHz signal strength is 6dB better than 433MHz, 9dB better than 902MHz, and 19dB better than 2400MHz at the same distance point. At the same receive power level, -120dBm, the 151MHz, 433MHz, 902MHz, and 2400MHz frequencies can achieve distances of 1.46km, 0.672km, 0.537km, and 0.198km, respectively for low-rate systems with sensitivities in the range of -110 to -120 dBm. Finally, note that the received signal power experienced a step decrease around 50m because of

an indoor/outdoor boundary for all frequencies, but the step is higher at 2400 MHz, which is consistent with diffraction effect theory.

CHAPTER 4 Energy Harvesting Radio System PHY/MAC layer Considerations

IEEE802.15.4 is defined as a standard for wireless sensor networks, but there has not been a defined technique for energy harvesting which demands an extremely low power supply stipulation. In Chapter 2, it was demonstrated that a burst communication, solar cell supported energy harvesting radio demo board requires average power consumption of 1mw or less. In Chapter 3, the choice of operating frequency based on propagation issues has been analyzed. Wesaw that the use of lower frequency benefits energy constrained radio system by as much as 20 dB, which translates to an energy advantage of up to a factor of 100. Lower frequencies were shown to propagate up to a radius of 1km with a 10mW transmit burst for low data-rate applications. This is much further than the 10m propagation radius defined within the IEEE802.15.4. However, the demo board lacks a protocol stack and only implemented an energy harvesting power transmitter. As we demonstrate in this chapter, it is impossible to apply the IEEE802.15.4 physical layer (PHY) and medium access control (MAC) layer synchronization techniques to this burst communication system directly. This is due to the constraints placed on energy consumption by the harvesting application. A more efficient physical layer frame format design would aid in this problem.

This chapter proposes VHF/UHF EHR system physical layer and medium access control layer synchronization techniques. The hardware validation is based on the existing K-State Micro-transceiver demo board (Figure 4-1), which offers the opportunity to prototype suitable receiveing functionality to complement the transmitter demonstrations of Chapter 2 and 3. The reason for the non-applicability of IEEE802.15.4 is prooven by analyzing its PHY and MAC layer definition in sections 4.2 and 4.3, and a new PHY layer specification and MAC synchronization method are then proposed. **Appendix A** gives additional information on the hardware and software used in the validation of these methods.

Figure 4-1 K-State Micro-Transceiver Demo Board

4.1 IEEE802.15.4 & ZigBee Overview

4.1.1 PHY General Definition

The IEEE802.15.4 standard defines the physical layer and medium access control sublayer specifications for low-data-rate wireless connectivity with fixed, portable, and moving devices that have limited power consumption requirements. ZigBee, the commercial name of this standard defined by the ZigBee Alliance, defines all layers, including network and application layers of IEEE802.15.4. Mesh networks with both star and peer to peer topologies are typical implementations.

The PHY layer defines two frequency bands, 2.4GHz and 868/915 MHz. Both bands use direct sequence spread spectrum (DSSS) modulation. Specifications of these frequency bands and data rate are shown within Figure 4-2.

PHY (MHz)	Frequency band (MHz)	Spreading parameters		Data parameters		
		Chip rate (kchip/s)	Modulation	Bit rate (kb/s)	Symbol rate (ksymbol/s)	Symbols
868/915	868–868.6	300	BPSK	20	20	Binary
	902–928	600	BPSK	40	40	Binary
868/915 (optional)	868–868.6	400	ASK	250	12.5	20-bit PSSS
	902–928	1600	ASK	250	50	5-bit PSSS
868/915 (optional)	868–868.6	400	O-QPSK	100	25	16-ary Orthogonal
	902–928	1000	O-QPSK	250	62.5	16-ary Orthogonal
2450	2400–2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal

Figure 4-2 Frequency band and Data rate of IEEE802.15.4 (IEEE802.15.4 2006)

ZigBee supports peer-to-peer and star network topologies. Two main kinds of devices are contained in the network, *coordinator* and *network device*. A "Beacon" frame transmitted by coordinators periodically is used to setup synchronization of the networking. There are two transmitting (TX) and receiving (RX) modes of a "Beacon-enabled" network shown in Figure 4-3. One is uplink and the other is downlink. Since, in one ZigBee network, the devices are not operating (TX / RX) continuously, they are working on active and inactive mode alternately with a specific duty cycle definition.

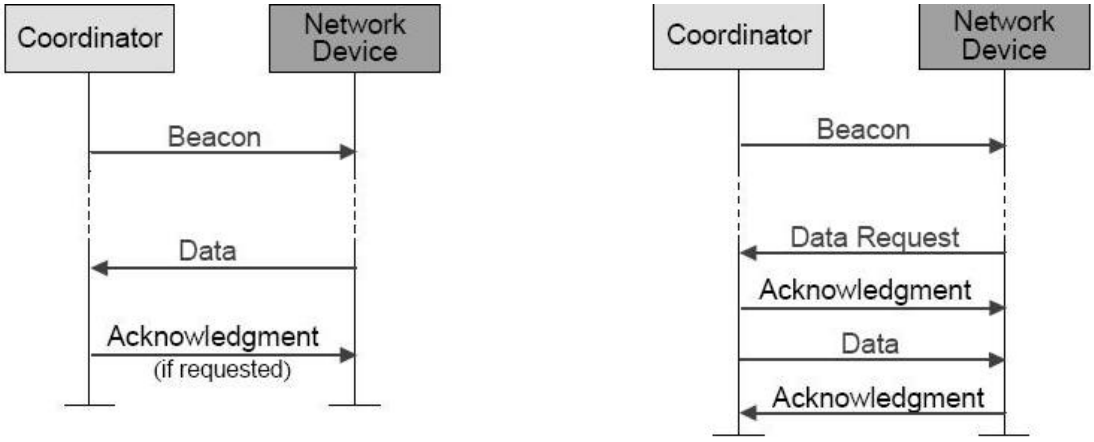


Figure 4-3 ZigBee beacon-enabled TX / RX mode [1]

4.1.2 MAC Layer Synchronization

A SuperFrame structure (Figure 4-4) is defined in IEEE802.15.4 bounded by the transmission of a beacon frame and can have an active portion and an inactive portion [1]. In

Figure 4-4, there are several variables, which are used to define the length of the superframe. *SD* is the active superframe duration. *BI* is the beacon frame interval. The variable *aBaseSuperframeDuration* is the number of symbols forming a superframe when the *SO*

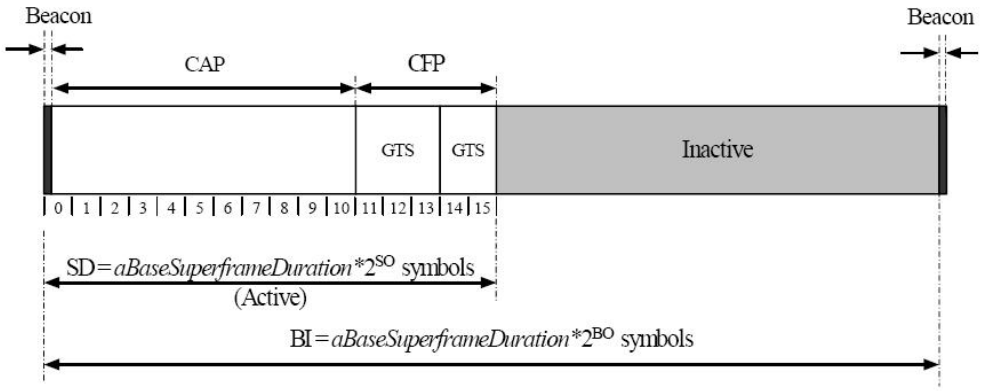


Figure 4-4 IEEE 802.15.4 Superframe structure [1]

(superframe order) is equal to 0. This variable is a constant, 960 symbols, equal to the product of the number of slots contained in any superframe (*aNumSuperframeSlots* is 16) and the number of symbols forming a superframe slot (*aBaseSlotDuration* is 60). The variable *BO* is the beacon frame order, which decides the duty cycle (the ratio of the active period to inactive period). Figure 4-5 shows the different duty cycle frames when the variable *BO* is defined as different numbers.

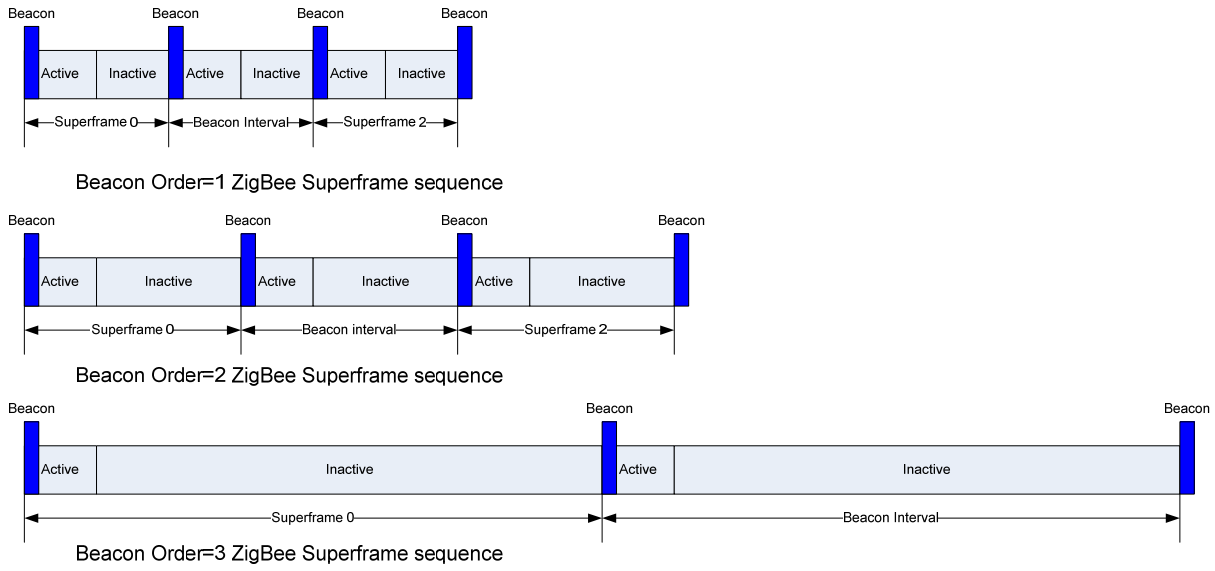


Figure 4-5 IEEE802.15.4 super frame sequence with different Beacon Oder

In IEEE802.15.4 (2006) section 7.5.4, it defines the MAC synchronization method: “To acquire beacon synchronization, a device shall enable its receiver and search for at most

[***aBaseSuperframeDuration*(2^n+1)***] symbols, where ***n*** is the value of *macBeaconOrder* (*BO*) [1].” Thus, depending on the data rate, the minimum (when *SO*=0, a superframe only has one active portion) active time, inactive time and synchronization time can be calculated by using equation (4.1), (4.2) and (4.3).

$$ActivePortionTime = aBaseSuperframeDuration / DataRate = 960 symbols / DataRate \quad (4.1)$$

$$InactivePortionTime = (aBaseSuperframeDuration \times 2^{BO}) / DataRate - ActivePortionTime \quad (4.2)$$

$$MaxSyncTime = aBaseSuperframeDuration \times (2^{BO} + 1) / DataRate \quad (4.3)$$

Table 4-1 shows the calculation results when *BO* equals from 1 to 10 by using 868MHz, BPSK, 20kbps PHY channel and 2.4GHz, O-QPSK, 250kbps (1symbol=4bits) PHY channel. Figure 4-6 shows sync time versus the duty cycle plots based on the Table 4-1 results.

Table 4-1 IEEE802.15.4 Synchronization time calculation

Frequency and Time		Beacon Order=1 DutyCycle50%	Beacon Order=2 DutyCycle25%	Beacon Order=3 DutyCycle11%	Beacon Order=4 DutyCycle6.25%	Beacon Order=5 DutyCycle3.13%
868MHz, BPSK, 20kbps	Active Time (s)	0.048	0.048	0.048	0.048	0.048
	Inactive Time(s)	0.048	0.144	0.336	0.72	1.488
	SuperFrameLen(s)	0.096	0.192	0.384	0.768	1.536
	Sync Time(s)	0.144	0.250	0.432	0.816	1.584
2.4GHz, O-QPSK, 250kbps	Active Time(s)	0.0038	0.0038	0.0038	0.0038	0.0038
	Inactive Time(s)	0.0038	0.0115	0.027	0.0576	0.1190
	SuperFrameLen(s)	0.0077	0.0153	0.0307	0.0614	0.1228
	Sync Time(s)	0.012	0.0191	0.035	0.0652	0.1266
Frequency and Time		Beacon Order=6 DutyCycle1.6%	Beacon Order=7 DutyCycle0.78%	Beacon Order=8 DutyCycle0.36%	Beacon Order=9 DutyCycle0.18%	BeaconOrder=10 DutyCycle0.09%
868MHz, BPSK, 20kbps	Active Time (s)	0.048	0.048	0.048	0.048	0.048
	Inactive Time(s)	3.024	6.048	12.096	24.53	49.10
	SuperFrameLen(s)	3.072	6.144	12.288	24.578	49.15
	Sync Time(s)	3.120	6.192	12.336	24.626	49.2
2.4GHz, O-QPSK, 250kbps	Active Time(s)	0.0038	0.0038	0.0038	0.0038	0.0038
	Inactive Time(s)	0.2419	0.488	0.9792	1.9622	3.928
	SuperFrameLen(s)	0.2457	0.536	0.9830	1.9660	3.932
	Sync Time(s)	0.2495	0.584	0.9868	1.9698	3.96

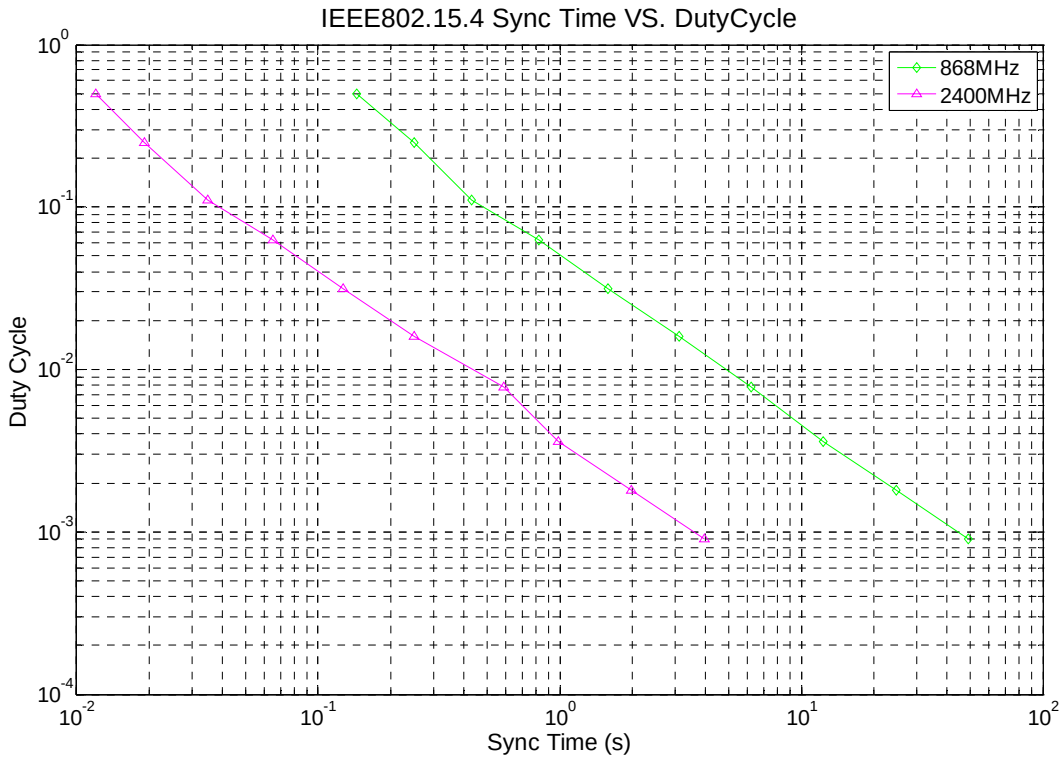


Figure 4-6 IEEE802.15.4 Sync Time Vs. Duty Cycle

From these results, we notice that it will take a substantial time to sync when the duty cycle is very low. From Table 4-1, when the duty cycle is 0.09%, it takes 49.2 seconds when working at 868MHz and it takes 3.96 seconds when working at 2.4GHz. This is not practical for a wireless sensor network, in which all coordinators, RFD and FFD devices are energy harvesting radios. We call this kind of network pure-EHR systems. This is because the radio has to stay on long enough to receive the beacon, assuming they do not yet know the network timing. For example, K-State's energy harvesting demo board has a duty cycle of 1.2% with a 30ms active portion and a 2.3s inactive portion. There are four 220uF capacitors used to save harvested energy from solar cells. Thus, if active RX period is 3.96 seconds, using the equation (2.3), the discharged voltage will be $dv = \frac{20.5mA \times 3960ms}{4 \times 220uF} = 92V$. This is impossible to be implemented on EHR system. A new MAC layer synchronization method therefore needs be designed for pure-EHR system that it is discussed in section 4.3.

4.2 Proposed EHR System PHY Layer

In this section, we propose a new method for synchronization which is applicable to highly energy-constrained network nodes. The proposed EHR's PHY layer solves the problem discussed above, including energy efficiency and synchronization energy consumption.

4.2.1 PHY General Definition

The assumptions used for PHY specifications of this application are:

- Central operation frequency (*aCentralRFFeq*) : 433.92MHz
- Modulation : FM wideband modulation with ±10kHz bandwidth
- Bit rate (*aMaxBitRate*) : ≤ 20kHz
- Maximum physical protocol frame length (*aMaxPHYFSize*) ≤ $BitRate \times ActivePeriod$, while the *ActivePeriod* is the active time period of burst communication system.

The central operation frequency (*aCentralRFFeq*) was selected as the VHF/UHF frequency band based on propagation measurements in the previous chapter. The central operation frequency of 433.92MHz delivers good transmission range performance.

The modulation method was selected to use FSK due to its faster synchronization property.

The bit rate is selected for a long range energy harvesting radio system. It is defined by the variable *aMaxBitRate*.

The maximum physical frame length is decided by this variable. For example, when *aMaxBitRate* is 1 kbps, and *aActivePeriod* is 30ms (as seen in chapter 2), the maximum physical layer frame length is 30 bits, and the maximum frame length is 150 bits if *aMaxBitRate* is 5 kbps. These rates are consistent with the -110 to -120 dBm sensitivity levels assumed in the measurements of Chapter 3.

The preamble and start of frame (SOF) parameters are significant parts of PHY protocol frame. They provide bit and frame synchronization for PHY layer, and decide the sync time. As we will show, the length of these parameters affects both PHY frame length and energy efficiency of EHR systems.

4.2.2 PHY Frame Structure

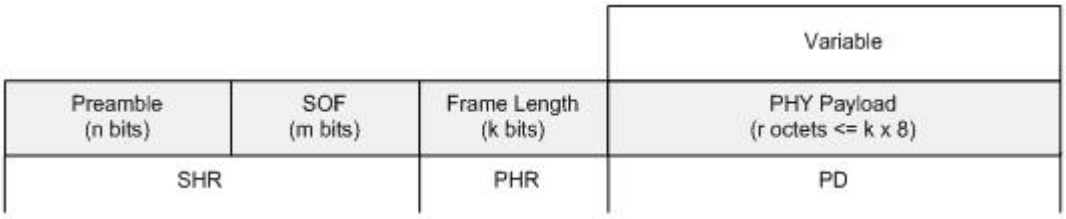


Figure 4-7 PHY Frame Format

Figure 4-7 defines the PROPOSED PHY frame format for EHR systems.

- Preamble: The Preamble field is used by the receiver to obtain bit synchronization with an incoming RF signal. The length of preamble, defined by the *aPreambleLength* variable, indicates how many bits are included in the preamble field. In section 4.2.3, we discusse the preamble design for EHR systems.
- SOF: The start of the frame is used to indicate the end of the preamble training sequence and the start of the message packet. The length is defined by the *aSOFLength* variable and the sequence is defined by the *aSOFSequence* variable. The Barker code, in Table 4-2, has lower autocorrelation properties and it is good to be used as the EHR start of frame sequence. The *aSOFSequence* "1101" has been used in measurements later because of its short active period.

Table 4-2 Barker code

aSOFLength	*aSOFSequence(Barker code)*
1	0
2	0 1
3	0 0 1
4	0 0 1 0
5	0 0 0 1 0
7	0 0 0 1 1 0 1
11	0 0 0 1 1 1 0 1 1 0 1
13	0 0 0 0 0 1 1 0 0 1 0 1 0

- Frame Length: The Frame Length field is 4 bits in length and specifies the total number of octets (1 octet = 8 bits) contained in the PHY payload. It has a value between 0 and *aMaxPHYPayloadSize* ($0 \leq aMaxPHYPayloadSize \leq 2^4$).
- PHY Payload: This is a variable length containing the data of PHY packet.

The frame length field is defined as 4 bits because of the burst communication type and low bit rate considerations. The energy harvesting prototype demo board of Chapter 2 was designed with a 30ms burst communication active period.

The PHY payload size could be larger, for example 10 octets (80 bits). There are two methods, which help to achieve this goal: (1) increase the active period time; (2) increase the transmitting data rate. The active transmitting period time can be increased by utilizing larger energy storage capacitors. The active capacitor discharging period has increased so the capacitor charging period must be increased too. For example, if the current active period (30ms) was increased to 300ms, with a data rate of 1 kHz, 300 bits or roughly 37 octets could be transmitted during a single duty cycle. However, the recharge time then increases too from 2.3 seconds to approximately ½ minute. This tradeoff introduces another dimension of the energy harvesting systems.

4.2.3 Preamble Design

For an EHR with low data rate and highly constrained on-time, minimizing the preamble length is critical to maximizing data volume that can be transmitted. This section will discuss the preamble length design and validation measurements. The minimum preamble length is determining by the shortest time that the receiver can bit-sync with the RF receiving signal. The demodulation, bit-sync, and DSP software used in experimental validation are described in Appendix A.

4.2.3.1 Preamble Length versus Start of Frame Detecting Error

The experimental validation environment includes a waveform generator, a signal generator, K-State's microtransceiver demo board and an oscilloscope. Figure 4-8 shows the experimental setup.

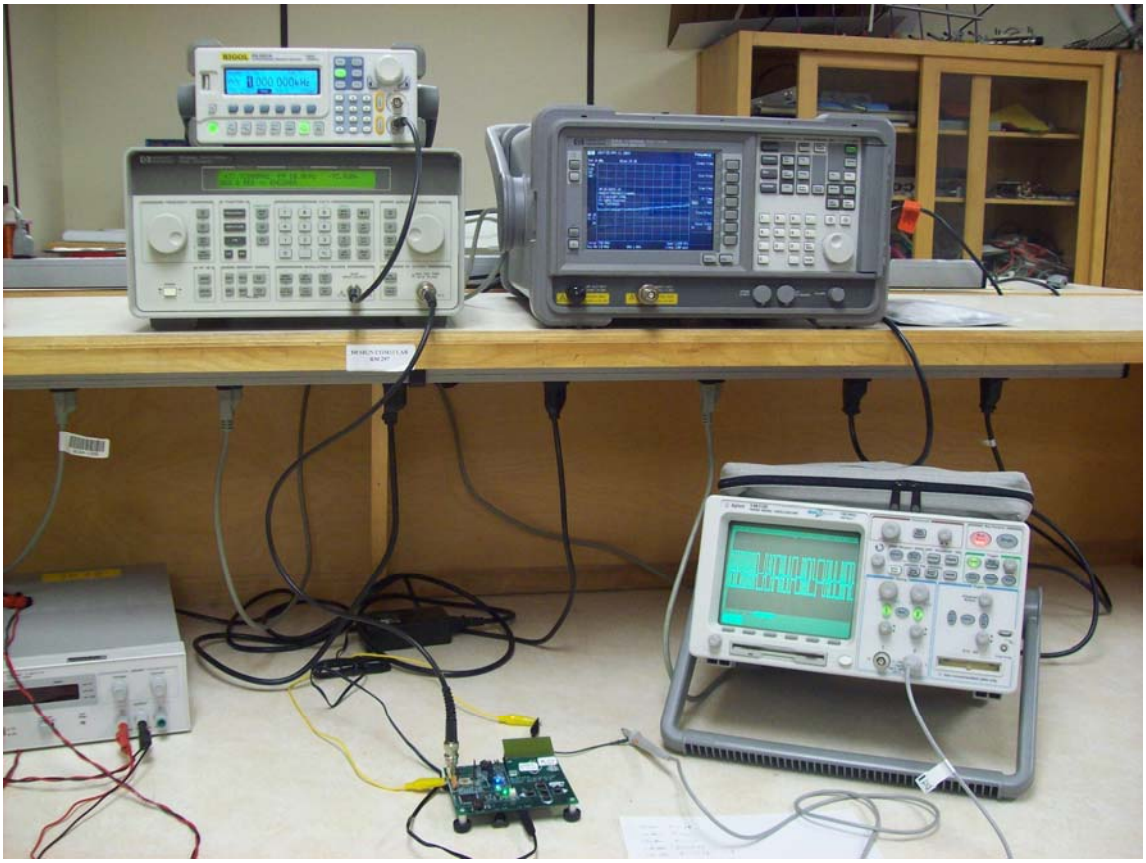


Figure 4-8 Preamble test experiment environment

The PHY frame was constructed using a waveform generator, which can create arbitrary bit sequences. Nine different types of frames were constructed using 6, 10, 20, 24, 28, 40, 80, 90 and 100 bit preamble lengths. The RF signal power range of -90dbm to -108dbm was set by signal generator. Figure 4-9 shows the resulting error detection probability of SOF field versus RF signal power of varied preamble length curves. Note that the receiver (described in Appendix A) employed a 40 kHz IF filter, which is about 10 times larger than a matched filter for the assumed data rate. Hence, a 10 dB better sensitivity can be achieved than that shown in Figure 4-9 at a given error rate. Note also that values and error-rate curve slopes above power levels of -98 dBm are only estimates, since the number of trials at each data point was limited (< 100).

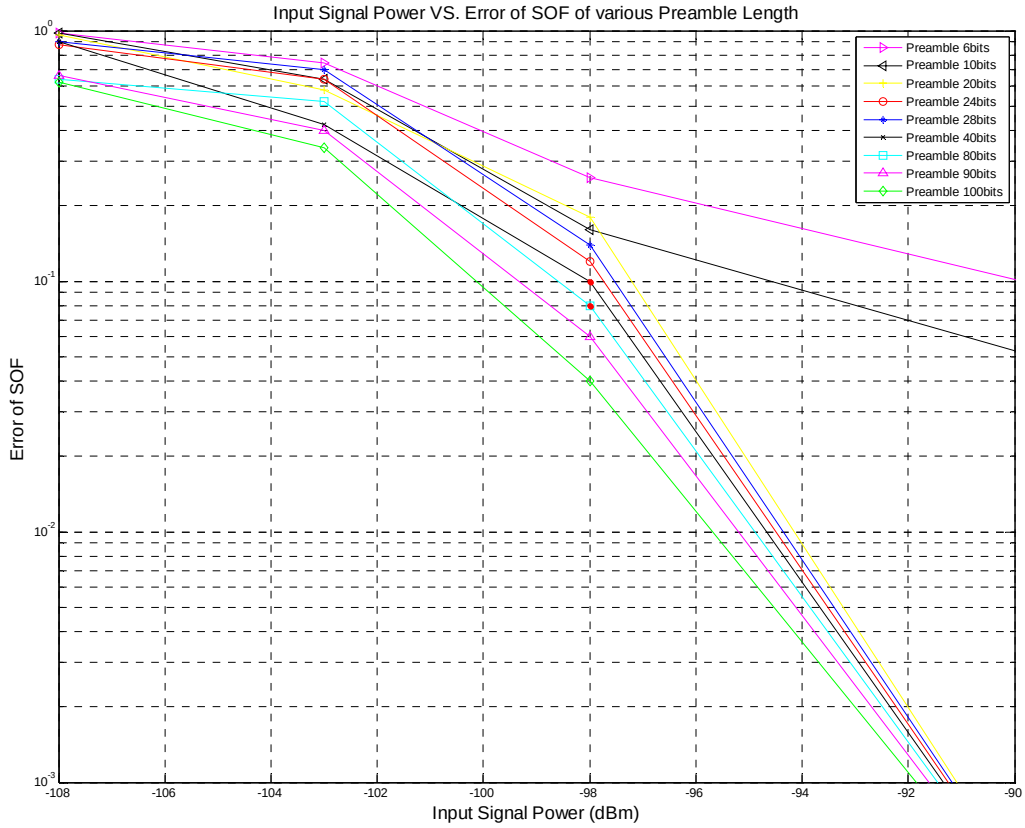


Figure 4-9 Error Probability of SOF versus Input Signal Power of varied preamble length

Figure 4-9 shows the following phenomenon:

1) To keep same error level, shorter preamble frames need more transmission power.
2) The error probability is very high when the preamble length is less than 10bits.

These results are collected by embedding a test flag within the bit-sync code. A pulse from the microcontroller I/O pin was generated when the test code detected a correct SOF sequence from incoming signal. Meanwhile, the demodulated and recovered bit sequence was output through an additional I/O pin. These two pins were probed and plotted using an oscilloscope. Figure 4-10 is screen capture of 30bits, 40bits, 50bits and 60bits preamble frames SOF detection. Therefore, we could observe if the software detected the SOF in the right place. By observing the output, error probability of SOF versus input signal power curve could be plotted.

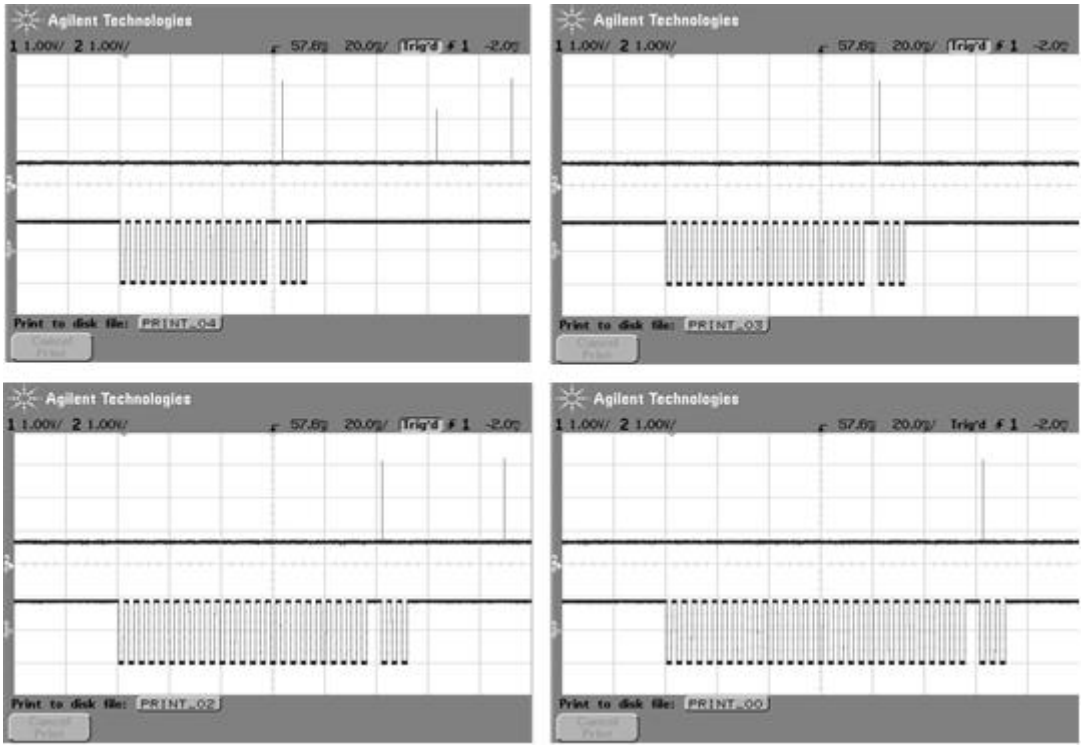


Figure 4-10 SOF detection with various preamble length

4.2.3.2 Method for Improving Short Preamble Frame Performance

The slope of 6 bits and 10 bits preamble curve in Figure 4-9 could be increased by using dynamic bit-sync lock window calculation or adaptive TX power method. Additionally, the transmit power could be decreased by a decrease in receiver bandwidth as previously mentioned.

Dynamic bit-sync lock window is different with the fixed bit-sync lock window. The largest lock loop window of a state machine could be used at very beginning synchronization processing. The large the lock loop window, the greater increase of bit-sync speed, but this will decrease the data recover accuracy. In practice, once a "0101" sequence is detected, shrinking the lock loop window to a proper size will increase the data recover accuracy..

Adaptive TX power involves transmitting more power during preamble period, but transmitting less power for the PHY payload part from the transmitter end. This will aid the receiver to sync the incoming bit stream with a short preamble sequence, but it may increase power consumption on the transmitter side.

The best way to increase receiver sensitivity is to decrease the receiver'snoise figure. For this demo board, the noise figure could be decreased by 3 to 5dbm, which should shift the curve left by 3 to 5dbm as well.

These methods were not implemented in K-State EHR demo board because it beyond of the research.

4.2.3.3 Preamble Length versus Energy Comsumption

Although Figure 4-8 has been shown the basic information of error probability of start of frame sequence versus transmission power consumption, the energy consumption should be more direct viewing for energy harvesting design. Table 4-3 shows calculation result of energy consumption of various preamble length frames based on the Figure 4-7 testing results.

Table 4-3 Preamble Length VS. Energy when data rate is 512bps

Preamble Len (bits)	SOF Len (bits)	Payload Len (bits)	P_RX (dbm)	P_TX (dbm)	P_TX (mw)	TX_Per (s)	TX_Energy (J)
6	4	72	-90	18	63.09	0.1602	10.11e-3
10	4	72	-94.61	13.39	21.83	0.1680	3.67e-3
20	4	72	-97.22	10.78	11.96	0.1875	2.24e-3
28	4	72	-97.54	10.46	11.12	0.2031	2.26e-3
40	4	72	-98	10	10	0.2266	2.27-e3

In Table 4-3, Payload Len item indicates that the test frames use same physical payload size 72 bits.

SOF Len item indicates that the start of frame field of PHY frame is 4 bits (‘1101’ sequence is used).

P_RX (dbm) is received power in *dbm* unit, which was power level fed into receiver antenna. These values were the X axis value when the *Error Of SOF* value of Y axis were same in Figure 4-20. Here *Error Of SOF* is 10^{-1}.

P_TX(dbm) is transmitted power in dbm unit left from transmitter antenna. Assume 10mw power is used to transmit 40bits preamble length frame. The difference power in dbm unit is $\Delta P\_RX = P\_RX(x) - P\_RX(40)$ where x corresping to preamble length. Thus, the P_TX(dbm) could be calculated by adding this ΔP_RX on 10dbm.

P_TX(mw) is transmitted power in mw unit left from transmitter antenna. This value is nothing but convert the dbm into mw unit.

TX_Per is transmitting active period in second unit. Since the data rate is 512bps for this test, the total frame length divided by 512bps gave out the transmitting time. For example, the

preamble length is10 bits, SOF is 4 bits and PHY payload is 72 bits, the transmitting period is $\frac{(10+4+72)bits}{512bps} = 0.168s$.

TX_Energy is transmitting energy used at transmitter side to send *(PreambleLen + SOFLen + PHYPayloadLen)* length frame. This energy is calculated by *TX_Per(s)*P_TX(mw)*.

Figure 4-11 shows the Energy consumption (mJ) versus Preamble Length (bits). This plot shows that a preamble length of 20 bits has good performance for its lower energy consumption.

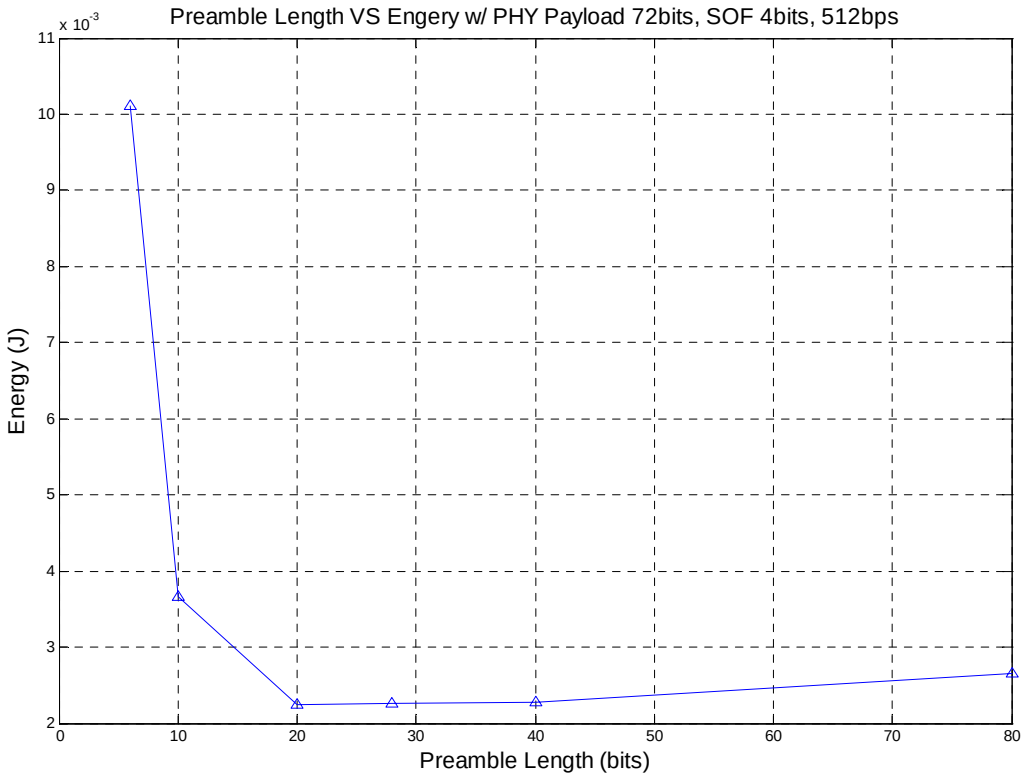


Figure 4-11 Energy Consumption VS. Preamble Length

4.3 Proposed EHR MAC Layer Synchronization

In previous senction 4.1.2, we have found that the IEEE802.15.4 defined synchronization method is not practical for pure-EHR systems, in which all corrodinators, RFD and FFD devices are energy harvesting radios. Thus, a new method for lower duty cycle EHR systems should be proposed. However, the WSN could be a hyrib-EHR system, in which some corrodinators are not energy harvesting radio but a power supplied node. With the reliable power supply, the synronization method will be different.

4.3.1 Pure-EHR Network Obtaining Synchronization

In the pure-EHR network, all WSN node devices are energy harvesting radio. Same as IEEE802.15.4 standard networking topologist definition, the energy harvesting radio systems have both star and peer-to-peer network topologies. The EHR network follows the IEEE802.15.4 definition, so there are PAN coordinator, coordinator, FFD, and RFD devices in the network.

Same as IEEE802.15.4, the superframe concept is used to define the different duty cycle (Figure 4-3 and Figure 4-4). The beacon frame is also be used for synchronization, which is transmitted from PAN coordinator periodically. The structure of beacon frame is shown in Figure 4-12, which start at the beginning of the active portion of a superframe within and occupy less than one slot duration. The beacon frame holds the beacon sequence number, which is represented by *aBeaconSequenceNumber* variable. This *aBeaconSequenceNumber* variable's value lies between 0 and *aMaxBeaconSequence*, which is related to the system duty cycle. The field BeaconControl contains the beacon's control bits to indicate this is a beacon frame.

Figure 4-12 Beacon Frame format

The synchronization of the EHR systems is designed different from IEEE802.15.4 standard since the receiver can only be on for very brief period. The device in IEEE802.15.4 network is active to scan beacon frame as most [***aBaseSuperframeDuration*(2^n+1)***] symbols. But the EHR couldn't work in an active mode for that long duration especially when the duty cycle is low (*n* is big). Since the EHR is communicating in burst mode, a *burst receiving method is designed to sync*. The sync processes are:

1. The EH receiver will receive at least 2 slots time duration data when power is on. Decode the received data to judge if the beacon frame is detected. If the beacon frame was not detected, the receiver will enter inactive mode to *sleep a defined inactive duration*.
2. When the EH receiver *first time wake up*, if the beacon frame was not detected, the receiver will enter inactive mode to *sleep a defined inactive duration plus one slot duration*.

3. When the EH receiver *second time wake up*, if the beacon frame still was not detected, the receiver will enter inactive mode to *sleep a defined inactive duration plus two slots duration*.
4. Therefore, with the increase of the receiver waking up, the sleep duration will increase one slot each time. The sleep duration will be reset when the waking up time touch the maximum number of slots of a Beacon Interval or a beacon frame is found.
5. Once the EH receiver detect the beacon frame, it will reset the local timer so that it could sync with the remote Coordinator using a suitable low powerd accurate timer. The keeping synchronization method is beyond of this research that it will not be discussed here.

The Figure 4-13 shows the process 2, 3 and 4 to obtain the synronization.

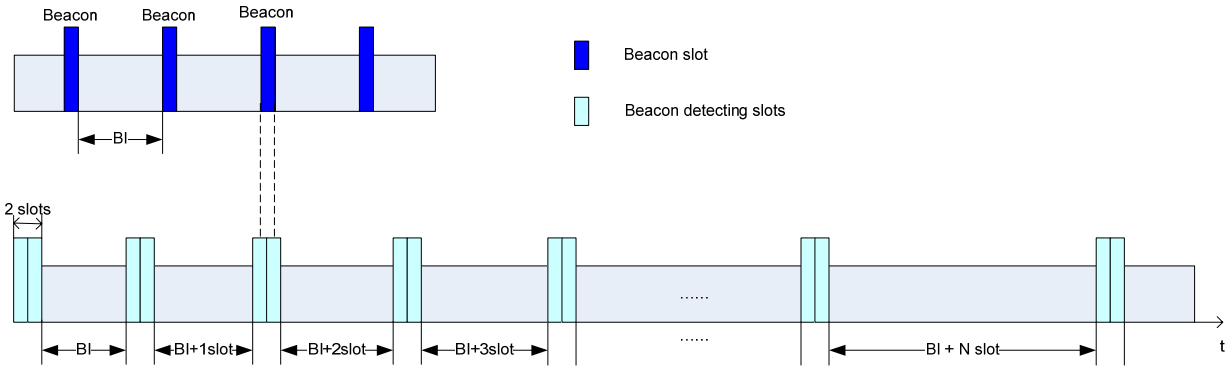


Figure 4-13 Beacon searching process

The Figure 4-14 shows the process 5 that a beacon frame has been detected.

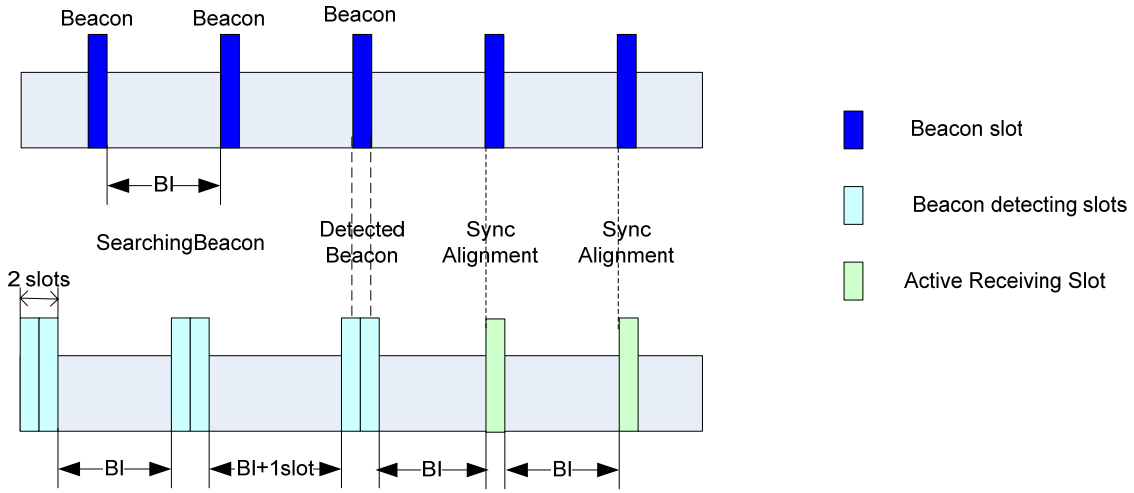


Figure 4-14 Beacon detected process

Here is an example of synchronization timing. The each time sleep duration could be expressed as ($T_{BI} + i*T_{SLOT}$), where T_{BI} is the time interval of two beacon frame, T_{SLOT} is time duration of one slot, and i is from 0 to the maximum slots number of one beacon interval duration. When use 1.3% duty cycle $P_{DC} = 0.013$ with 30ms active duariton T_{ACT}, and 2.3s inactive duration T_{INACT} defined in K-State EH demo board, the number of slots in one beacon interval duration $K_{SLT} = 777$, which is calculated by using equation (4.4).

$$K_{SLT} = \frac{T_{BI}}{T_{SLOT}} = \frac{T_{ACT} + T_{INACT}}{T_{SLOT}} \tag{4.4}$$

The relationship between duty cycle, active duration, inactive duration and beacon interval is expressed by equation (4.5).

$$T_{BI} = T_{ACT} + T_{INACT} = T_{ACT} + \frac{T_{ACT}(1 - P_{DC})}{P_{DC}} \tag{4.5}$$

The maximum synchronization time $T_{maxSYNC} = 1757s \approx 30min$, is calculated by using equation.

$$T_{\max SYNC} = \sum_{i=0}^{K_{SLT}} (T_{BI} + i \times T_{SLOT}) + T_{SLOT} \tag{4.6}$$

Obviously, the different duty cycle leads to different synchronization durations since K_{SLT} is changing with duty cycle as well as the T_{BI}. Table 4-4 shows the duty cycle versus synchronization durations using K-State's EHR prototype and using IEEE802.15.4 syncronization method. Since the K-State EHR board duty cycle is designed for $P_{DC} \leq 1.3$ (Table 2-1) and the active duration is 30ms for low CPU clock design, 5 duty cycles between 0.09%~1.6% were selected to compare with IEEE802.15.4.

Table 4-4 Sync time and power consumption comparison of EHR and IEEE802.15.4 standard radio

Duty cycle	EH demo board		IEEE802.15.4
	Sync time	Active duration voltage drop (v) *	Active duration voltage drop (v) *
1.6%	30 minutes	0.7	72.7
0.78%	123 minutes	0.7	144.2
0.36%	9.6 hours	0.7	286.5
0.18%	39 hours	0.7	573.1

0.09%	6.4 days	0.7	1146.4

(*The active duration voltage drop is calculated by using equation (2.3) and assumes active/inactive current are 20.5mA and 0.06mA. These number came from Chapter 2, Duty Cycle Design, section "Low CPU Clock, Hardware control TX")

For IEEE802.15.4, none of the voltage drops are possible; they are shown only to emphasize the need for the proposed new synchronization technique.

Figure 4-15 is the plot of EHR sync time versus duty cycle.

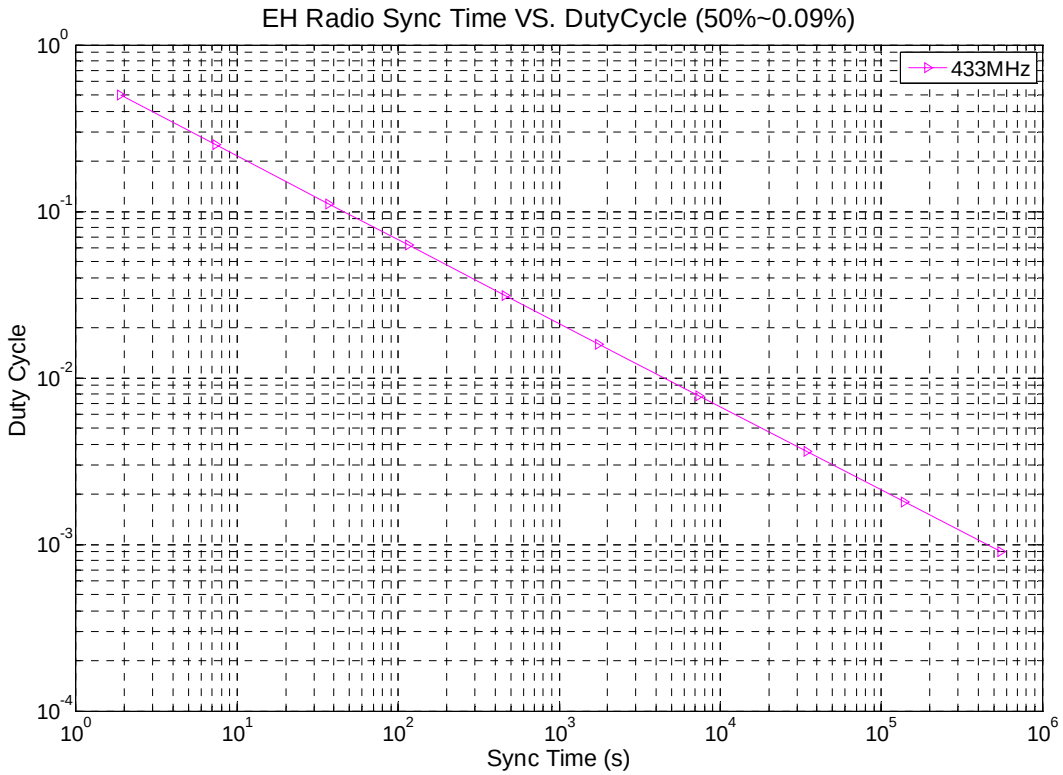


Figure 4-15 Energy Harvesting Radio Sync time versus Duty cycle

4.3.2 Hybrid-EHR Networks for Obtaining Synchronization

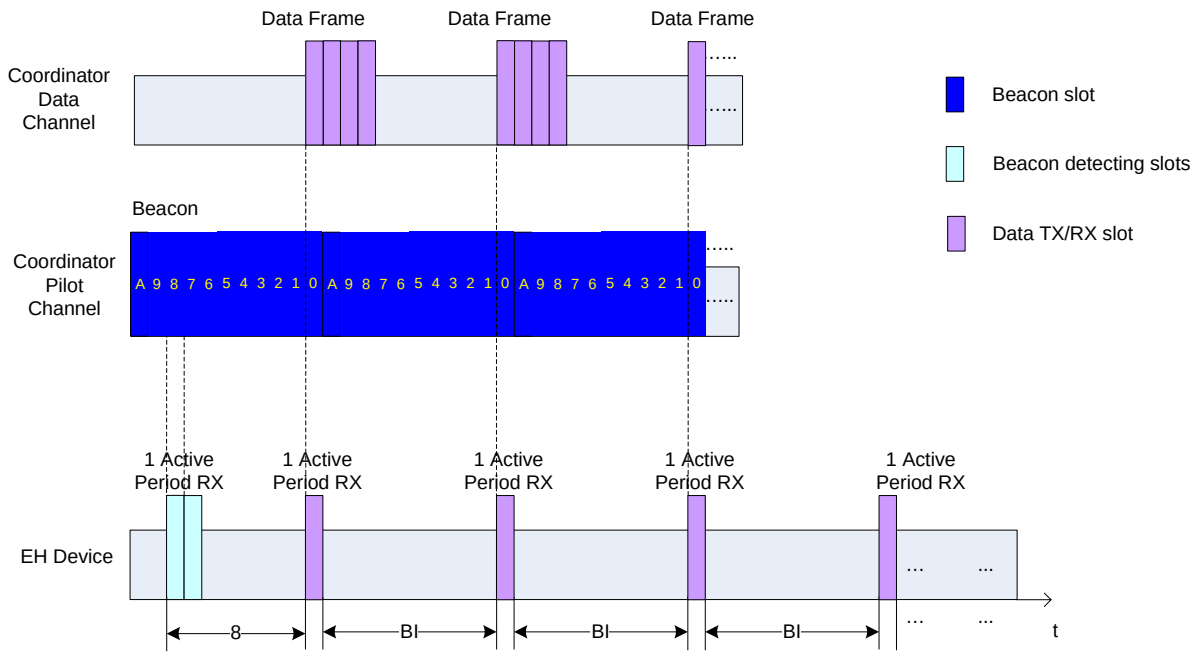
The problem was found from the Table 4-4 that the synchronization period becomes over 10 hours when the duty cycle is less than 0.36%. This will happen especially when the energy harvesting source is weak so that the device has to sleep longer to recharge capacitors. Thus, a hybrid-EHR netowork is proposed, in which some coordinators are not energy harvesting radios but have higher power supplies. That means the coordinator can transmit beacon frames in a continuous mode instead of the burst mode without energy concerns. Hence, synchronization can occur more quickly for the nodes in the network that use energy harvesting.

There are two considerations. One is assuming we have enough spectrum resource so that a specific pilot channel can be set up to transmit beacon frames specifically (Figure 4-16(a)).

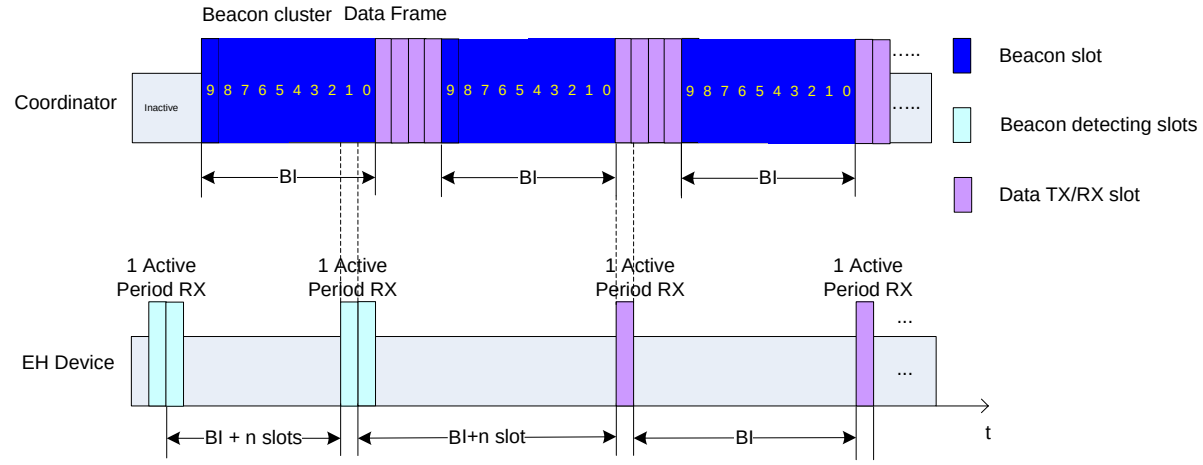
1. The WSN coordinator continually transmits beacon frame clusters using a pilot channel (only for synchronization accessing without data transmission);
2. The beacon frame cluster contains multiple numbers of beacon frames. Each beacon frame includes the synchronization information (eg. Cycled beacon frame number 255~0) to indicate the left time from the next data frame transmitting or receiving time moment;
3. The EH devices of the WSN wakes up to receive the beacon frame on the pilot channel periodically with its defined duty cycle as in the pure-EHR systems definition. It shifts one slot each time and once it has detected one beacon frame, the EH device will jump to the data channel and adjust the local timer depending on the synchronization information packaged inside of the beacon frame. It will then sleep until the next data frame transmitting or receiving time moment;

The other case is the limited spectrum resource environment. In this case the beacon frame and the data frame must share the same channel. Thus the beacon frame and the data frame are transmitted alternately in one channel (Figure 4-16(b)).

1. The coordinators transmit the beason frame cluster periodically but with a high duty cycle design (since the coordinators are not EH powered). The beacon frame cluster is alternated with active data frames during active period of the duty cycle;
2. The EH devices of the EHR wake up to receive the beacon frame as with the pure-EHR network and shift one slot each time until find the synchronization



(a) Multi-channel network



(b) One channel network

Figure 4-16 Hybrid-EHR System Synchronization

Thus, when the EHR systems work at ultra-low duty cycle while the powered nodes operate at higher duty cycles. The EH devices will find synchronization immediately when the pilot channel is used, and the maximum synchronization time will be reduced to a couple of duty cycle durations when the beacon and data frame share one channel. For example, when the duty cycle is 0.005%, the beacon interval ***BI*** is 600s≈10min. If the powered coordinator transmits continuous beacon frame clusters in the pilot channel, it will find the synchronization

immediately. If the coordinator transmits periodically beacon frame clusters and data frames in same channel with a 50% duty cycle, it will take at as much as 2****BI***≈20min to find the synchronization. In either case, the time will be substantially reduced from the pure EHR network design and should be useable for most EHR applications..

4.3.3 Keeping Synchronization

To keeping synchronization is an important function of the MAC layer. The keeping synchronization is that once the EHR node obtained the synchronization, it should periodically check and adjust local timer to keep synchronization with the networking. However, we don't propose any technique on this area, we only list several popular techniques, which have been used for wireless sensor network applications. The fundamentals technique key words are listed below the technique.

- NTP : Network Time Protocol
  - Used in internet
  - Local sync to NTP server
- RBS : Reference Broadcast [21]
  - Reference message is broadcast (Receiver-Receiver)
- TPSN : Timing-sync protocol for sensor networks [22]
- FTSP : Flooding time synchronization protocol [23]
  - MAC layer time stamping
  - Used by Crossbow (a ZigBee products company)

For EHR systems, keeping synchronization design could reference these techniques but it difinitly can not use them directly. This will be a future research direction for EHR systems.

4.4 Conclusions

In this chapter, we overviewed the IEEE802.15.4 standard for wireless sensor networks, including the PHY layer frame structure and the MAC layer schronization processes. Since this standard is not defined for battery-free WSN nodes, it is impossible to use it as PHY and MAC layer definition for energy harvesting radio sytems. For an EHR system with low data rates and highly constrained on-time, minimizing the PHY preamble length is critical to maximizing data volumes that can be transmitted. Thus, a preamble design was discussed and validated by using K-State Micro-transceiver demo board and EHR DSP software. Another part of this chapter covered MAC layer synchronization design. Since an EHR uses capacitors as energy storage to support the burst communication, the IEEE802.15.4 sync-process would lead to impractical

voltage drops if it was used for EHR system. Thus, we proposed MAC synchronization for EHR systems. The technique may be used in either pure EHR networks where all nodes, including the coordinator, are subject to EH power constraints. Alternatively, for faster synchronization, a hybrid technique, where coordinator nodes have higher energy sources, can be considered.

CHAPTER 5 Conclusions

5.1 Summary

Current research in the wireless sensor network area has focused mainly on smart energy management, effective synchronization algorithms, and energy efficient network protocols. An industry wireless sensor network standard, IEEE802.15.4, is defined to serve a set of industrial, residential and medical applications with a short transmission range (<10m) and battery supply operation in the 868/915MHz or 2.4GHz frequency bands.

However, the desire for the future wireless sensor network application s includes battery-free nodes, extended RF range, and self-network organization. To satisfy these desires, the challenges of energy harvesting power source operation, RF communication modes, RF link budgets and propagation, and synchronization techniques should be considered simultaneously.

In this thesis, we proposed solutions for energy harvesting radio systems including their PHY layer frame structure and MAC layer synchronization techniques. Firstly, we developed an energy harvesting radio (EHR) prototype demo board using indoor solar cells as the energy harvesting source. This is a bettary-free, burst communication mode radio node. The radio uses 433MHz as its RF operation frequency, and the average current is less than 0.32mA. Secondly, four VHF/UHF frequency's propagation, 151MHz, 433MHz, 902MHz, and 2.4GHz, were measured using different antenna techniques, demonstrating that the lower frequencies can transmit further and have a smaller path-lost exponent than the higher frequencies typically used. The transmission range at 10 mW transmit power can extend up to 1 km using 151MHz, and 0.7 km using 433MHz from indoor to outdoor with a directional antenna at one end and monopole antenna at the other. Finally, two proposed MAC layer synchronization techniques were discussed for both pure-EHR and hybrid-EHR systems. The IEEE802.15.4 defined synchronization method cannot be used for the low duty cycle energy harvesting regime since the synchronization duration is too long to be supported by the energy captured in reasonable sized capacitors. The proposed synchronization method works at ultra-low duty cycle (<2%) along with burst communication mode.

5.2 Challenges and Future Directions

Although the prototype energy harvesting radio system and its PHY/MAC consideration discussed in this thesis is a good solution for battery-free, long distance, and low-cost WSN, there are still several challenges and future research directions that need to be mentioned.

The most important challenge is low speed interactive communication. When using EHR nodes, the duty cycle has to be kept low to maintain average power consumption at sub-milliWatt levels. Especially when the energy harvesting sources are weak, the duty cycle will be very low. For example, if the device awakes once per hour, transmitting four data frames will require four hours. Thus, a compact application protocol layer is required to solve this challenge.

For EHR PHY/MAC layer implementations, keeping timeslot synchronization will be one of the most important points. New techniques to keep synchronization under ultra-low power consumption constraints are needed. In addition, future work is needed to design a network layer and higher application layer for an EHR system. . Some related work has recently begun under a standardization effort known as 802.15.4f. It is hoped that this thesis will contribute to this effort.

Bibliography

[1] IEEE Computer Society, "IEEE Std 802.15.4TM-2006", New York, NY, 2006

[2] Simjee, F., Chou, P.H, "Everlast:Long-life, Supercapacitor-operated Wireless Sensor Node", Lower Power Electronics and Design, 2006.

[3] Hang Su, Xi Zhang, "Battery-Aware TDMA Scheduling Schemes for Wireless Sensor Networks", In Global Telecommunications Conference, 2008.

[4] J.Alberola, J.Pelegri, R. Lajara, Juan J. Perez, **"**Solar Inexhaustible Power Source for Wireless Sensor Node", In IEEE International Instrumentation and Measurement Technology Conference (FMTC)*,* 2008.

[5] Eliasson, J, Lindgren, P., Delsing, J., Tompson, S.J, Yi-Bing Cheng, "A Power Management Architecture for Sensor Nodes", In Wireless Communications and Networking Conference, 2007

[6] Youssef, M., Yousif, A., EI-Sheimy, N., Noureldin, A, "A Novel Earthquake Warning System Based on Virtual MIMO-Wireless Sensor Networks", In Electrical and Computer Engineering Canadian Conference, 2007.

[7] D'Souza, M., Bialkowski, K.,Postula, A., Ros,M, "A Wireless Sensor Node Architecture Using Remote Power Charging, for Interaction Applications", Digital System Design Architectures, Methods and Tools, 2007

[8] K.Daniel Wong, "Physical Layer Considerations for Wireless Sensor Networks", In International Conference on Networking, Sensing & Control IEEE*,* 2004

[9] Raviraj, P., Sharif, H., Hempel, M., Song Ci, "MOBMAC – an energy efficient and low latency MAC for mobile wireless sensor networks", Systems Communications, 2005

[10] William Kuhn, Jeongmin Jeon, and Kai Wong, "A Low-Power, Radiation-tolerant, RFIC Micro-Transceiver Chipset for Space Applications", NASA VLSI Symposium, 2007

[11] Bill Kuhn, **"**White Paper on Energy Harvesting Radio Transceivers", 2007

[12] John S. Seybold, "Introduction to RF Propagation", John Wiley & Sons, 2005

[13] Christopher Coleman, "An Introduction to Radio Frequency Engineering", The press Syndicate of the University of CAMBRIDGE, 2004

[14] W.L.Stutzman and G.A.Thiele, “Antenna Theory and Design, 2nd ed”, Wiley, Hoboken, 1998

[15] Kai Chang, **“**Handbook of Microwave and Optical Components”, Microwave passive & Antenna Components, Volume 1, 1997

[16] B.Sklar, “Digital Communications Fundamentals and Applications, 2nd ed”, Prentice-Hall, Upper Saddle River, 2001

[17] Foroozesh, A.; Shafal, L, “Size reductin of a microstrip antenna with dielectric superstrate using meta-materials: artificial magnetic conductors versus magneto-dielectrics”**,** In Antennas and Propagation Society International Symposium IEEE, 2006.

[18] Collins,S.; Antar, Y.M.M, “Antenna size reduction using non-planar rings”, Electronics Letters, Volume 38, Page(s):677-679, 2002

[19] Carol Wilson, “Frequency Bands for Wireless Sensor Networks”, CSIRO ICT Centern, Australia

[20] Ane Kristoffsersson, “Digital Switchover and Spectrum Dividend Market Status 2006”, *HiQ,* 2006

[21] J. Elson, L. Girod, and D. Estrin, “Fine-Grained Network Time Synchronization using Reference Broadcasts”, In Proceedings of the 5th symposium on Operating systems design and implementation, 2002

[22] S. Ganeriwal, M. Srivastava, “Timing-sync Protocol for Sensor Networks (TPSN) on Berkeley Motes NESL”, 2003

[23] Miklos Maroti, Branislav Kusy, Gyula Simon, Akos Ledeczi, “The Flooding Time Synchronization Protocol”, In Proceedings of 2nd International Conference on Embedded Networked Sensor Systems, 2004

[24] EnOcean Inc, “Dolphin system architecture for self-powered sensors and wireless sensor networks”.

[25] http://ieee802.org/15/pub/TG4f.html

Appedix A - PHY Layer implementation of K-State Energy Harvesting Receiver System

The whole energy harvesting system can be divided into two parts: the RFIC front-end and the DSP baseband. The RFIC front-end provides a physical data transmitting service. The DSP baseband provides modulation and MAC layer provides the sampling, matched filter, bit-sync and frame-sync services.

A.1 RFIC Front-end

The RFIC Front-end is provided by the K-State micro-transceiver. This micro-transceiver's specifications are:

1. Operating frequency (f_c) of 433.92Mhz (a unlicensed band)
2. IF frequency(f_{IF}) *of* 10.7MHz
3. IF filter bandwidth (BW_{IF}) *of* 40 kHz
4. FM frequency deviation (Δf) *of* ±10kHz
5. Receiver sensitivity ≥ -120dbm
6. Low current consumption (*Iop*) < 20mA during both TX and RX modes
7. Data rate (R_d or f_m) of 1 kbps

This system is designed as a Wideband FM modulation system and has implementing FSK modulation for this RFIC Front-end. Equation (A-1) is used to describe a Narrowbnad or Wideband FM system.

$$\beta = \frac{\Delta f}{f_m} \qquad \text{Equ(A-1)}$$

β less than one indicates Narrowband FM, whereas β greater than one indicates Wideband FM. The IF filter bandwidth (BW_{IF}) of 40 kHz is greater than the Wideband FM modulation bandwidth of 20 kHz, so, that the FM modulated signal is passed through the IF filter. Thus, to implement FSK in this RFIC front-end is possible. Figure A-1 show spectrum relationship between IF filter and Wideband FM modulation.

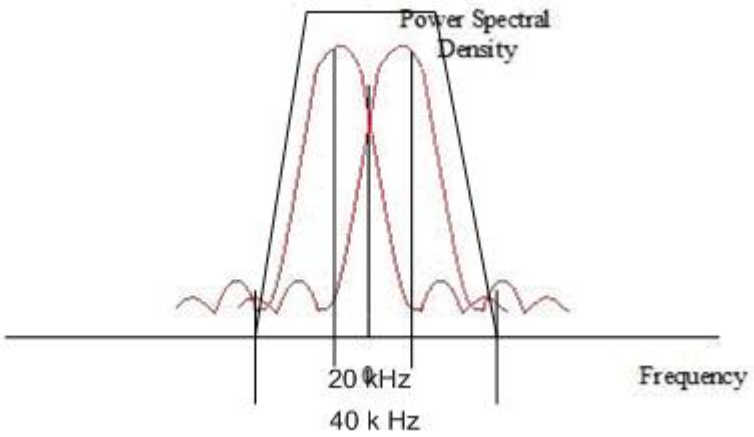


Figure 5-1 IF filter and wideband FM modulation

A discussion of typical FSK demodulation is needed prior to discussing a One-shot method. Typically, FSK demodulation has coherent and non-coherent methods. Figure A-2 shows block diagram of those demodulation methods. The correlator Receiver design (a) requires the receiver to generate the same phase and frequency carry wave *s(t)* to mix with

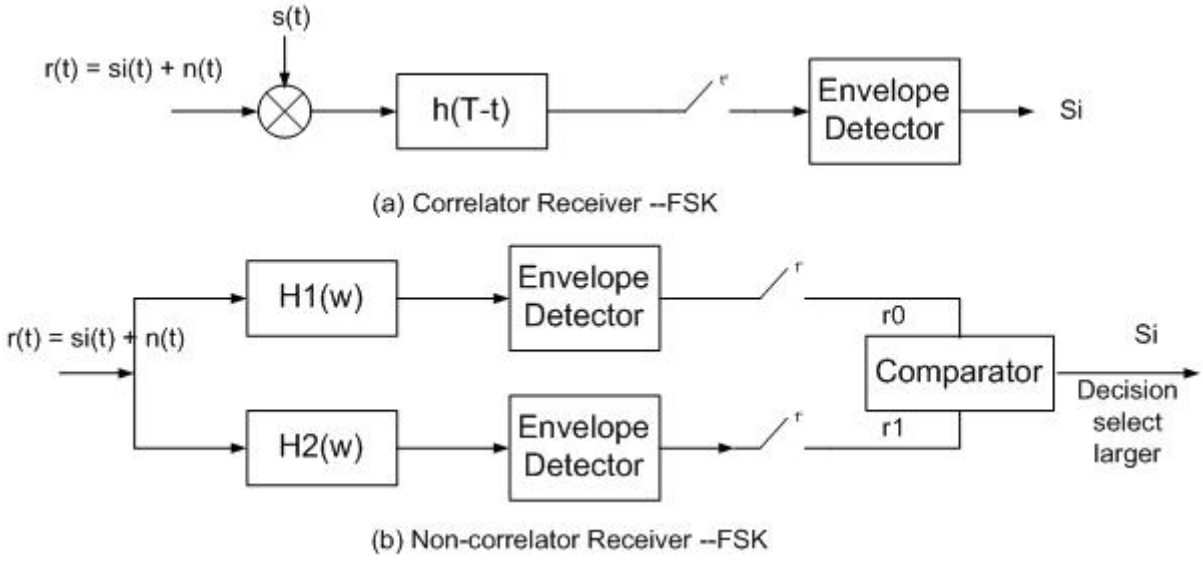


Figure 5-2 FSK demodulation (a) Correlator Receiver (b) Non-correlator Receiver

the received signal *r(t)*. Following the mixing operation is a matched filter *h(T-t)* that aids in providing maximum SNR at sampling instant time t_m. Once the signal is sampled a envelope detector will recover the sent digital signal.. The non-correlator receiver design (b)does not provide the same phase and frequency carry wave, but utilizes two matched band-pass filters, *H1(w)* and *H2(w),* to match the RF pulses corresponding to the digital signal. These two filters operate at the center frequency, $f_c \pm \Delta f$, with reasonable bandwidth (since the band pass filter

has ring-up and ring-down time ≈ 1/BW of the filter). The filters are followed by an envelope detector with an outputs at $t=T_b$. These outputs r_0 and r_1 have Gaussian noise components *n0* and *n1* (with a standard deviation of σ_n). The outputs are sent to a comparator, which will output a zero if $r_1 < r_0$ and output a one if $r_1 > r_0$. These two receivers can be implemented either in hardware or software in DSP, but high MIPS would be needed for the DSP option.

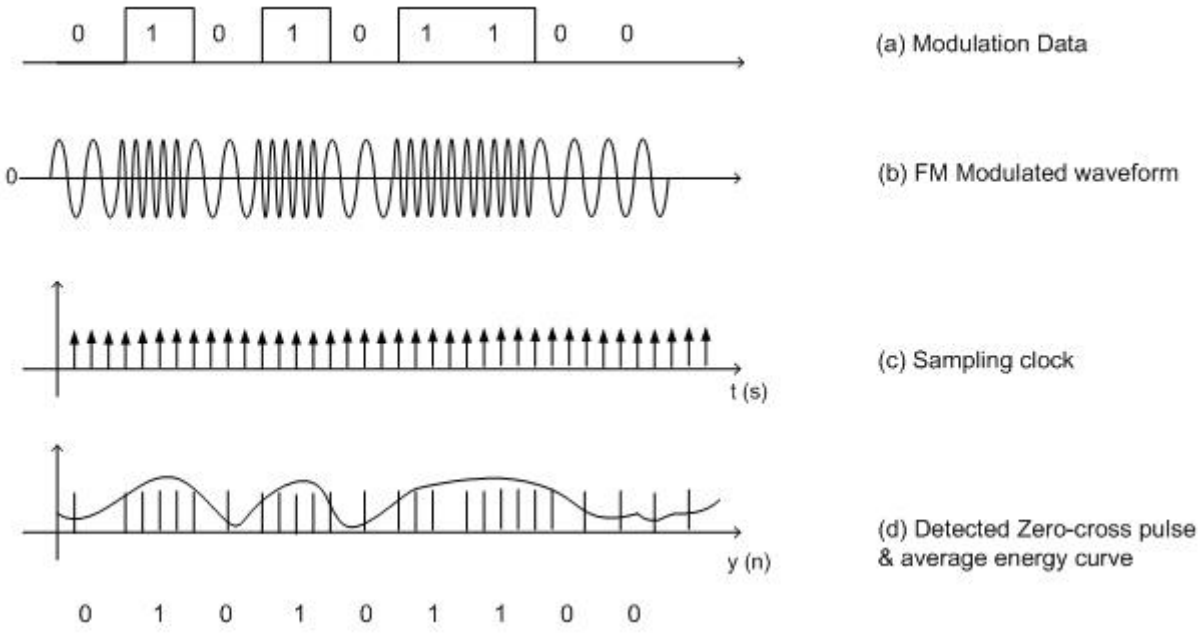


Figure 5-3 One-shot FSK demodulation: (a) Modulation Data, (b) FM Modulated waveform, (c) Zero cross Sampling clock, (d) Zero cross pulse & average energy curve

A One-shot FSK Demodulation will replace the previously mentioned methods based on an existing 16-bit microcontroller and K-State RFIC front-end. This technique is based on digital signal processing and statistical signal processing. Figure A-3 is diagram of this method. We called it "One-shot" since the K-State Micro-transceiver provides 1-bit ADC IF output. A one is outputted if the IF waveform is above threshold, otherwise a zero is outputted. A serial sequence denoted as $x_0,x_1,x_2\ldots x_n$ can be created by sampling the output of the 1-bit ADC. Detecting "zero crossings" allows for the detection of frequency changes and leads to recovery of the sent digital signal. "Zero crossings" imply that x_{n-1} is zero and x_n is one. A FM modulated signal varies the operation frequency up and down to denote the digital bit being sent. We assume that a zero is modulated to $f_c - \Delta f$ and one is modulated to $f_c + \Delta f$. A "zero crossing" denotes that a one is detected, while a zero is detected if there was not a zero crossing" in $y_0,y_1,y_2\ldots y_n$ serial

sequence. Thus ***Y(n)*** contains dense clusters of ones if high frequencies are sampled, alternatively thin clusters of ones if low frequencies are sampled. If think ***Y(n)*** serial as a pulse serial, to get average energy of it, we can easily get a waveform, which is similar with modulation data waveform. Figure 4-4 shows the One-shot demodulation technique using a"zero cross" detector.

The objective of Analog FM demodulation is to reproduce the original waveform. Digital communication systems have inputs that are sets of finite symbols (here they are either 0's or 1's). The objective of receiver is not to reproduce the waveform exactly but to decipher which symbol has been transmitted. The average energy level curve in Figure 4-4(d) reflects the input transmitted symbol with noise.

Now the issue is how to obtain the serial sequence ***Y(n)*** with a 16bit Microcontroller sampler. Subsampling and matched filtering techniques are solutions for this issue.

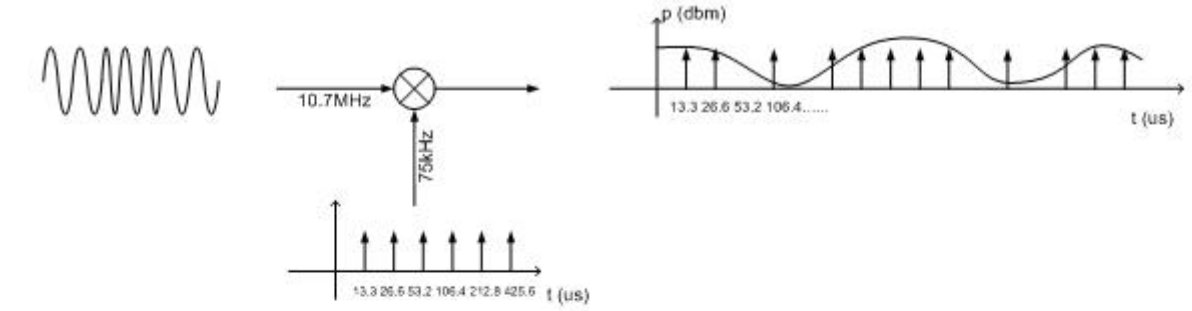
A.2 DSP Baseband Design

A.2.1 Subsampling

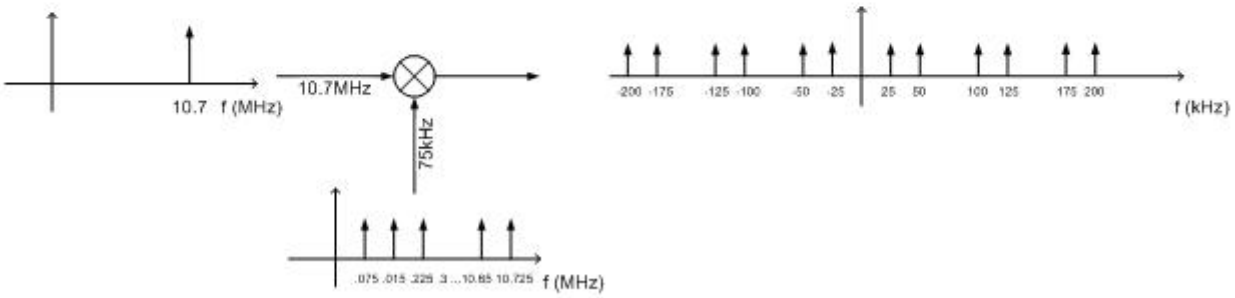
Nyquist-Shannon sampling theorem states that an analog signal that has been sampled can be perfectly reconstructed from the samples if the sampling rate is equal to or greater than 2****f*** samples per second, where ***f*** is the highest frequency of original signal. This is also called oversampling. The K-State RFIC operates at 433.92MHz and a Intermediate Frequency (IF) of 10.7MHz. Using the Nyquist theorem, we should use at least 2 × 433.93 = 867.86 MHz sampling rate or sample once every 1.15ns. It is impossible to implement this fast sampling rate on a 16bit, or 32bit low power consuming microcontroller. But, the Nyquist oversampling theorem is adequate for baseband signal. For a band limited pass band signal, the Nyquist sampling theorem can be reworked that if the sampling rate equal to or greater than 2****B*** samples per second, where ***B*** is the bandwidth of band pass signal. Again, the pass band analog signal can be perfectly reconstructed from the samples if the 2***B** sampling rate is observed. This is called subsampling. Subsampling only is adequate for band limited systems.

Base upon the above sampling theorem foundation, we designed a subsampling system for K-State 433.92MHz FM micro-transceiver front-end. Since the FM frequency deviation is ±10kHz, the modulated signal lies between 433.93 MHz and 433.91 MHz. that the corresponding IF frequency are between 10.69 MHz and 10.71 MHz with a pass band bandwidth

20 kHz. The sampling rate is designed at 75 kHz, which is much greater than 20 kHz bandwidth, to sample 10.7 MHz IF 1-bit ADC output. The front-end outputs a one if the IF frequency is above the threshold, otherwise it outputs a zero. A external 75 kHz sampling clock can be provided to front-end from the microcontroller.



(a) Subsampling 10.7MHz to 25kHz —Time domain



(b) Subsampling 10.7MHz to 25kHz —Frequency domain

Figure 5-4 Subsampling 10.7 MHz with 75kHz sampling rate

The important purpose of subsampling is to down convert the 10.7 MHz IF frequency to a baseband frequency which could be processed by 16bit microcontroller. This can be seen in both time-domain and frequency-domain representations. Figure A-4(a) shows the time-domain representation of the subsampled signal. We sample the 10.7 MHz FM signal at $n \times \frac{1}{75kHz} = n \times 13.3us(n = 1,2,3...)$. A output pulse indicates a zero crossing, while no pulse indicates no "zero crossing". Since the FM modulated signal is 20 kHz bandwidth compared with 75 kHz sampling rate we are still oversampling. We assume FM modulates a one using the high frequency of $f_m + 10kHz$ and a zero using the low frequency of $f_m - 10kHz$. Thus, in Figure A-4(a) we see dense pulse cluster samples indicating high frequency and sparsely pulse

samples indicate low frequency. If we acquire the average energy of these pulses, we will get a waveform with a threshold above zero.

FigureA-4(b) shows the frequency-domain representation of the subsampled signal. The 10.7 MHz 1-bit ADC IF is sampled at a sampling rate $f_s = n \times 75kHz(n = 1,2,3...)$ using impulses., These impulses mix the IF frequency with these $\boldsymbol{f_s}$ frequencies and it can be down convert or up convert to the center at $\boldsymbol{f_s}$. When $\boldsymbol{n}$ is equal to 143, $f_s = 10.725MHz$. When $\boldsymbol{f_s}$ is mixed to 10.7 MHz, it down converts the IF signal to be centered around ±25 kHz. When $\boldsymbol{n}$ is equal to 142, $f_s = 10.65MHz$. When this $\boldsymbol{f_s}$ mixes with 10.7 MHz, it down converts the IF signal to be center around ±50 kHz. The Figure A-4(b) shows all spectrums of mixed signals.

Thus, subsampling provides a good way to down convert higher pass band signals to lower baseband signals. This allows a 16-bit microcontroller to process the lower sampled signal competently and allows the one-shot FSK demodulation to be implemented. A new issue becomes how to recover the average energy level curve of ***Y(n)*** binary serial sequence? If a D/A convertor is not utilized, then the task is easy. Since we do NOT have an D/A in our PIC24 microcontroller the software method is only way to implement this. Therefore, a matched filter becomes the best way to solve this.

A.2.2 Digital Low Pass Matched Filter

A digital low pass matched filter (LPMF), with a 1 kHz cut off frequency, has two functions The first function is frequencies greater than 1 kHz are block. The second function is to match the 1 kHz data rate to recover the data pulse sequence with lower noise. Since the filter is filtering not only higher frequency signals, but also noise, it can increase the SNR of signal.

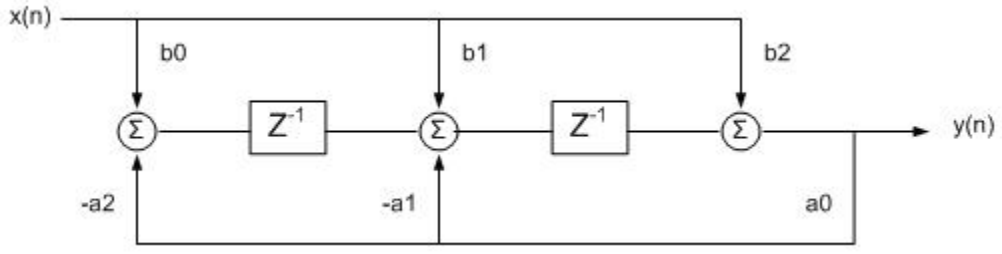


Figure 5-5 Two pole IIR digital low pass filter

In order to implement LPMF in real-time, a 2-pole IIR low pass filter is designed. This allows for low MIPS consumption and linear system stability.. The Figure A-5 is diagram of two

pole IIR filter. Filter coefficients dependent on the cut off frequency and sampling frequency are needed to for the filter design.

In S-domain, a 2-pole filter can be expressed as

$$H(s) = \frac{1}{s^2 + \sqrt{2} \times s + 1} \qquad \text{Equ (4-2)}$$

The cut off frequency is $f_c = 1kHz \rightarrow w_c = 2\pi f_c$

Thus $$H(\frac{s}{w_c}) = \frac{1}{(\frac{s}{w_c})^2 + \sqrt{2}(\frac{s}{w_c}) + 1} \qquad \text{Equ(4-3)}$$

For IIR filter, a bilinear transform is needed by replacing ***s*** with

$$s \approx \frac{2}{T_s}\frac{1 - z^{-1}}{1 + z^{-1}} = \frac{2}{T_s}\frac{z - 1}{z + 1} \text{ where } T_s = \frac{1}{75kHz} \text{ is the sampling time period} \qquad \text{Equ(4-5)}$$

So H(z) becomes $$H(z) = \frac{z^2 + 2z + 1}{[\frac{4}{(T_s w_c)^2} + \frac{2A_1}{T_s w_c} + 1]z^2 + [2 - \frac{8}{(T_s w_c)^2}]z + [\frac{4}{(T_s w_c)^2} - \frac{2A_1}{T_s w_c} + 1]} \qquad \text{Equ(4-6)}$$

where $A_1 = \sqrt{2}$

The IIR filter Z-domain function

$$H(z) = \frac{b_2 z^2 + b_1 z + b_0}{a_2 z^2 + a_1 z + a_0} = \frac{b_2 + b_1 z^{-1} + b_0 z^{-2}}{a_2 + a_1 z^{-1} + a_0 z^{-2}} = \frac{Y(z)}{X(z)} \qquad \text{Equ(4-7)}$$

The ***Y(z)*** output of filter is aset of ***y(n)***,***y(n-1)***,***y(n-2)*** *taps*. The***y(n)*** tap represent the current output and ***y(n-p)*** tap represents the past outputs. The ***X(z)*** *tap*, input of filter, is a set of ***x(n)***, ***x(n-1)***, ***x(n-2)*** *taps*. The ***x(n)*** tap represent the current input and ***x(n-p)*** tap represent the past inputs.

Thus $a_2 y(n) = b_2 x(n) + b_1 x(n-1) + b_0 x(n-2) - a_1 y(n-1) - a_0 y(n-2)$ Equ(4-8)

If we normalize a_2=1 and divide each coefficient by a_2, then we can get current output ***y(n)***

$$y(n) = b_2' x(n) + b_1' x(n-1) + b_0' x(n-2) - a_1' y(n-1) - a_0' y(n-2) \qquad \text{Equ(4-9)}$$

From Equ(4-6) we can calculate the filter's coefficients ***a'*** and ***b'***.

Since *Ts* = 13.3uS, $w_c = 2\pi f_c$=6.28 krad/s, The coefficients are shown in Table A-1:

Table A-1 Two pole LPF coefficients

a2=1	a1=-1.419	a0=0.5533
b2=0.0336	b1=0.0671	a0=0.0336

To verify that the coefficients are correct for the design, we used MATLAB to calculate the coefficients and to plot the filter response. This detail will be discussed in software implementation. Figure A-6 shows the frequency response comparison of the self-calculated coefficients and the MATLAB ***butter()*** function calculated coefficients.

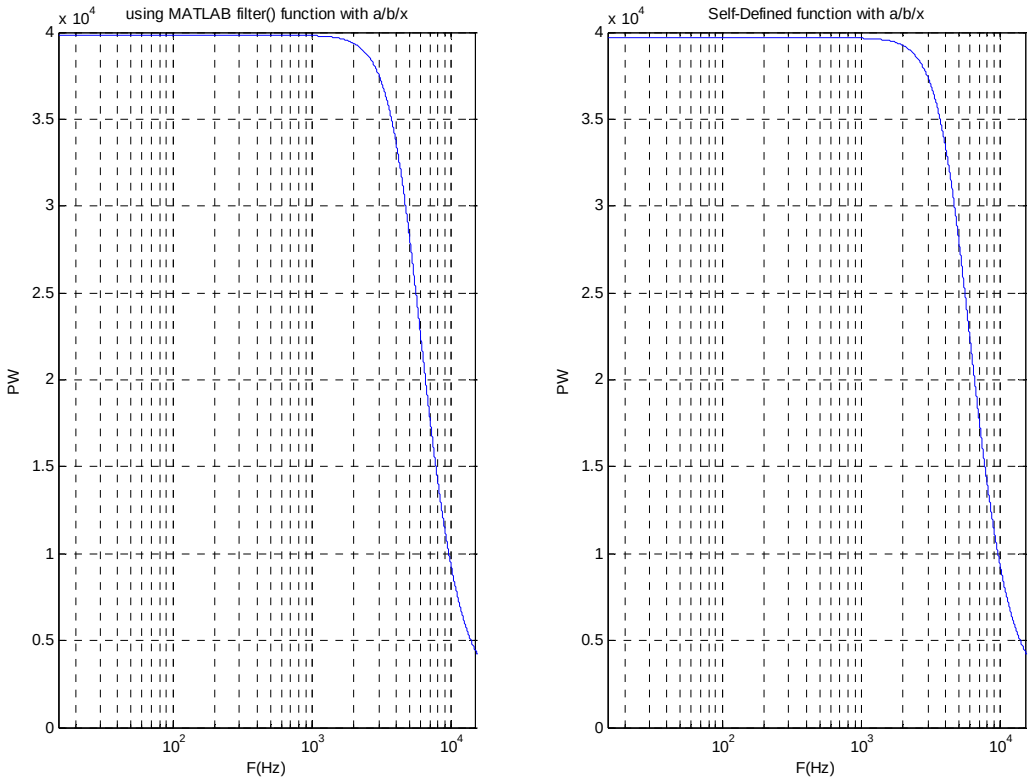


Figure 5-6 Frequency response comparison of designed 1 kHz Low Pass Matched Filter with MATLAB function plot

The input ***X(z)*** came from RF front-end output is either a 1 or 0. We map ***X(z)*** to -2^{14} and $+2^{14}$ before we put it into LPMF for three important reasons:

1. In order to divide 0 with DC voltage threshold
2. We use a 16-bits Microcontroller without floating operation. The highest bit is used to indicate sign. In the case of left shifting causing overflow, we only use 14 bits to save the data

3. We use shift (<<,>>) and plus (+) operations instead of the product operation (x) to implement float operation so that we can save CPU operating cycle and make it possible to implement on a low CPU clock

The ideal output of low pass matched filter are a points set of recovered FSK data sequences similar to the ones shown below. The levels are between -2^{14} and $+2^{14}$ representing 0 and 1 respectively. But, in the real world, since input sampling will be affected by noise, the output from RF front-end will be affected too. Figure A-7(a) shows the ideal and real world output of a low pass matched filter and (b) shows the real world output of a low pass matched filter.

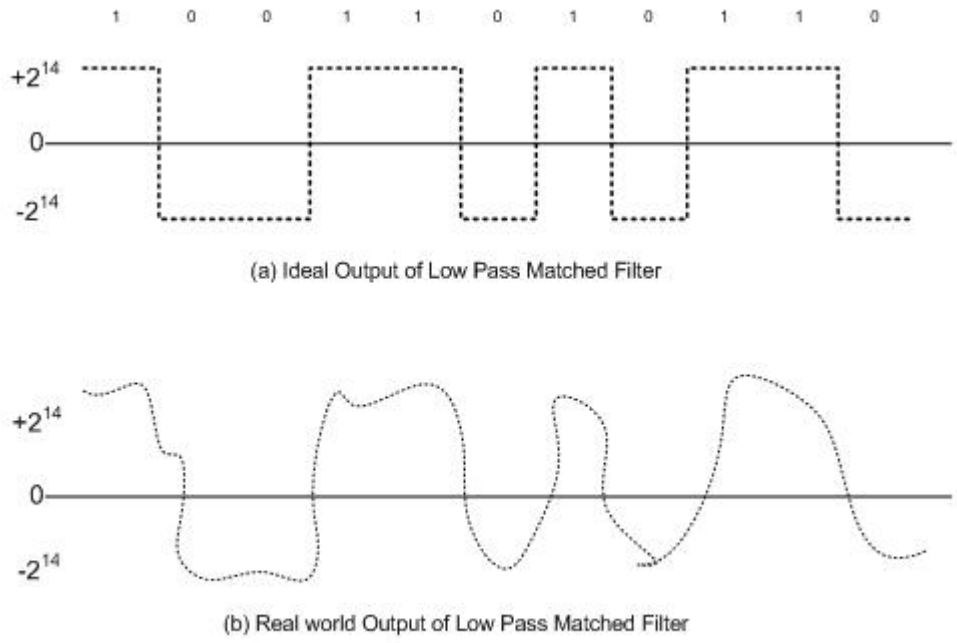


Figure 5-7 output of LPMF (a) Ideal output (b) real output

From Figure A-7, a new issue needs to be considered. Which points in the output of LPMF are the rising edge and which are falling edge? Which points are the middle of a pulse and where can we get the data we expect? To answer this question, we need to perform a Bit-sync. Bit-sync aids in the detectection of risinging edges and provides a phase locked clock with waveform. This allows the data to be sampled in the middle of each pulse.

A.2.3 Bit-sync

Bit-sync is providing bit synchronization with fixed data rate in our design. It tracks the present signal and aligns its local clock with the detected bit transition. It generates a phase locked clock and aligns the clock raising edge at the middle of each data pulse meanwhile which frequency can be shifted little by little if data pulse raising edge shifted. Figure A-8 is diagram of Bit-sync.

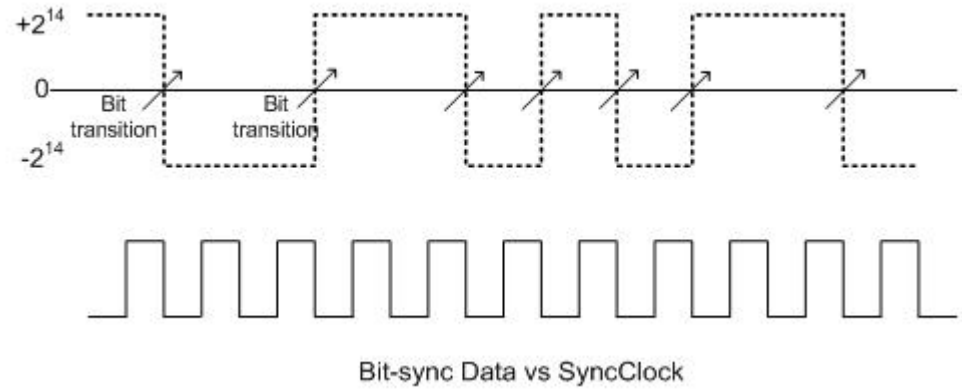


Figure 5-8 Bit-sync data versus clock

There are two main steps that need to be realized a bit-sync algorithm. The first step is detecting bit transitions in which the algorithm searches the bit transition or threshold crossing. The second step is aligning the data in which the algorithm is aligning a clock based on a Finite State Machine (FSM).

To detect a transition, we need to process the ***Y(n)*** serial sequence. ***Y(n)*** has a real-time output of the LPMF at 75 kHz. Since a 512 bps data rate is 200us / bit, there are almost 150 samples (75 kHz = 13.33us) during a single bit period. Thus, we do not need to detect ***Y(n)*** at the 75 kHz sampling rate, instead we can sample it at a much lower rate. In this design, we sample 1/3 of subsampling frequency, which is 25 kHz. That means, we have 50 samples of each bit and it is enough to detect bit transition. This also aids in reducing the microcontroller processing burden, saving MIPS.

A 50 state FSM is designed because the 25 kHz sampling frequency and 512 bps bit rate. The amount of FSM states can be calculated using equation (4-10). The BitPeriod is 1/512 and the BitSyncSamplingRate is 1/25kHz giving us 48.8 states.

$$FSM\# = \frac{BitPeriod}{BitSyncSamplingRate} \qquad \text{Equ(4-10)}$$

Figure A-9 shows the Bit-sync FSM design and state transfer path. Every 1/3 of the LPMF outputs will be put into the Bit-sync process and the output drives state transfer. The FSM starts as the state S0. The state transfers from S0 to S46 whenever it detects a bit transition or not. If a transition has occurred it should be recorded into a single transition flag variable. At state S24, which should be middle of a bit period, a sync-clock raising edge is generated. At state S46, the transition flag variable is checked to see if a transition occurred. If a transition had occured the state goes back to S0, otherwise it progresses to the next state.

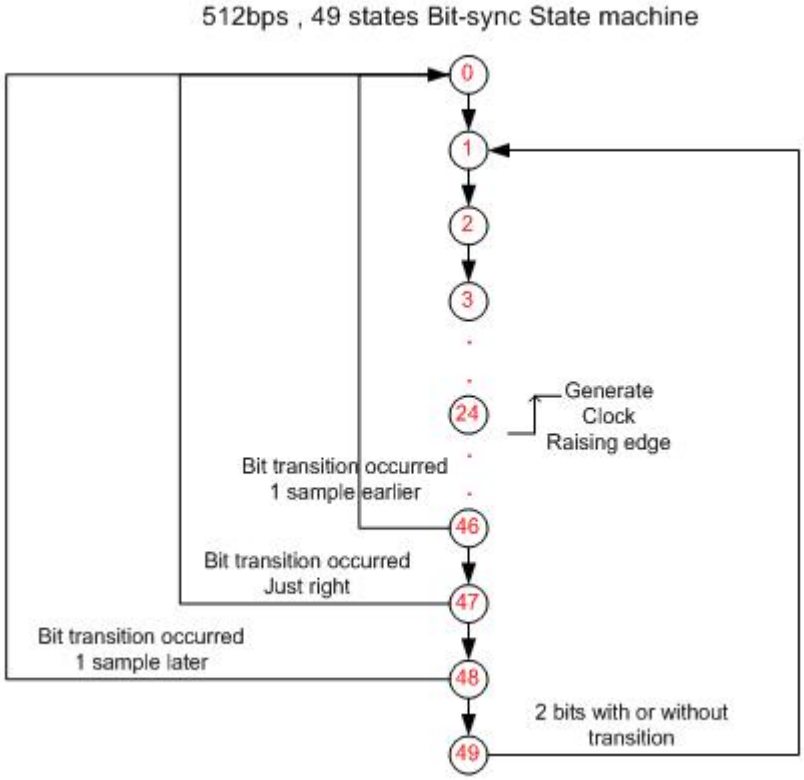


Figure 5-9 Bit-sync Finite State Machine (FSM)

Which means a transition happened before expected and the sync-clock needs to catch up to align the data. At state S47, the expected state to reset the state machine, if the transition occurred, state machine goes back to state S0, otherwise transfer to the next state. At state S48, 1 sampling later, if the transition occurred, state machine transfer back to state S0, otherwise transfer to the last state. At state S49, whatever happened before or at this state, transfer back to state S1. One situation is that the transition had occurred at this state, which is 2 samples later than expected. Going back to state S1 instead of state S0 will shorten the Sync-clock next rising edge by one sample. Another situation is that a transition did not occur during the whole

FSM process (S0 to S49). This generally implies that two of same bits were transmitted. Thus going back to state S1 instead of state S0 will shorten Sync-clock too. The second situation may cause sync-clock shifting after long repeated bit transmitting, but it will re-sync as long as the bit stream changed.

A.3 DSP Software Implementation

A.3.1 Microcontroller Configurations

All design theory has been addressed and now it is time to explain software implementation. The existing K-State Demo board hardware design block diagram is shown in Figure A-10. Microchip PIC24HJ256, high-performance, 16-Bit Microcontroller is employed as the center process unit.

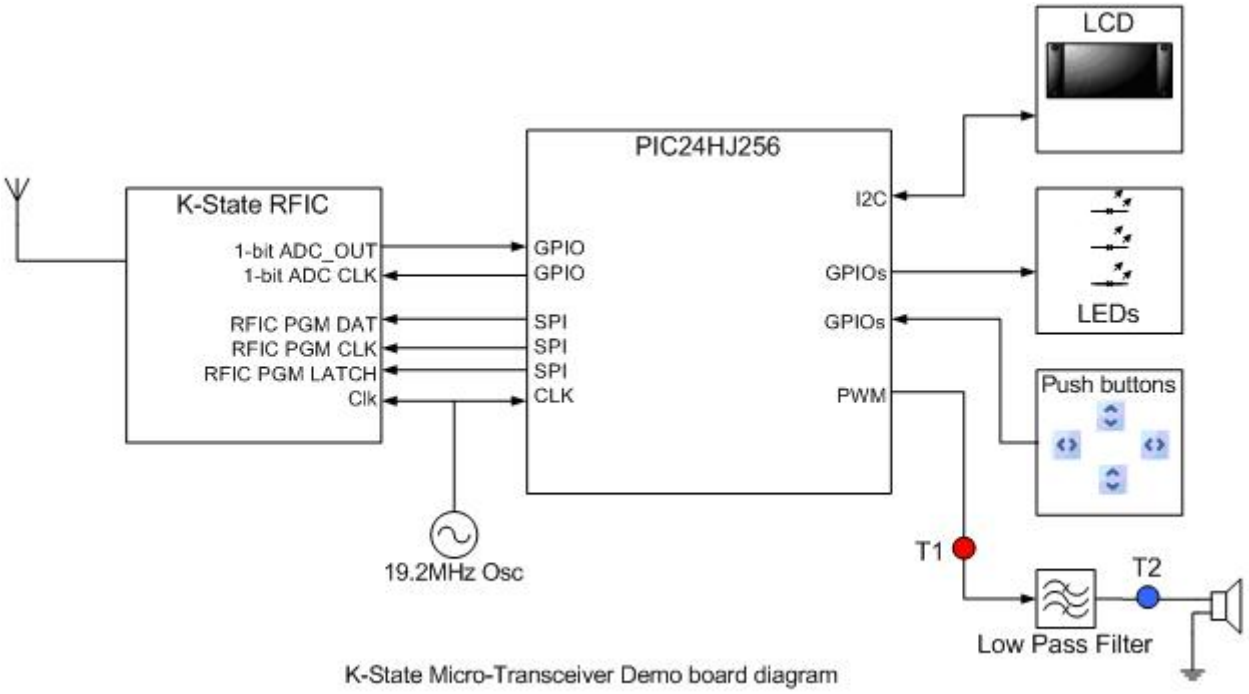


Figure 5-10 K-State Micro-Transceiver Demo board block diagram

It is NOT a physical DSP architecture, but the features on the microcontroller are adequate enough to implement the low data rate, FSK One-shot demodulation and Bit-sync DSP processes. The microcontroller runs on a 19.6MHz clock configuration and capable of running up to 9600000 instructions per second. The main features of this microcontroller are:

- Modified Harvard architecture

- C compiler optimized instruction set
- 16-bit wide data path
- 24-bit wide instructions
- Linear program memory addressing up to 4M instruction words
- Linear data memory addressing up to 64 Kbytes
- 256 Kbytes flash program memory and 16 Kbytes Data SRAM
- 71 base assembly instructions : mostly 1 word / 1cycle
- Sixteen 16-bit General Purpose Registers
- Software stack
- CMOS Flash Technology leads low-power consumption

It also provides interrupt controller and many peripherals:

- Interrupt Controller
    - 5-cycle latency
    - 118 interrupt vectors
- Digital I/O
    - 85 programmable digital I/O pins
    - Output pins can drive from 3.0v to 3.6v
    - 4mA sink on all I/O pins
- Timer/PWM
    - Nine 16-bit timers/counters or pair up to four 32-bit timers
    - Output compare provides 16-bit Glitchless PWM mode
- Communication Modules
    - 3-wire SPI—support 8/16-bit data
    - I2C—Full Multi-Master Slave mode,7/10-bit addressing, and bus collision detection
    - UART
    - Enhanced CAN
- System management
    - External, internal RC clock with $F_{CY} = \frac{F_{OSC}}{2}$

    - Power-up timer
    - Watchdog timer
    - Reset by multiple sources
- Power management
    - On-chip 2.5v voltage regulator
    - Switch clock sources in real time
    - Idle, sleep and Doze modes with fast wake-up

We are not using all of those features, but all of them are helpful to design low-power consumption EH demo board.

In this design the microcontroller is working at below configuration:

- F_{OSC} using 19.2 MHz External clock. Instruction cycle is 0.104us since $F_{CY} = \frac{F_{OSC}}{2}$ where F_{CY} is instruction execution speed and F_{OSC} is oscillator source speed
- 3.3v power supply
- Watchdog timer disable
- One SPI be used to program RFIC
- One I2C be used to program LCD
- 16bit Timer2 is used as 1-bit ADC sampling clock
- Digital GPIOs are used to connect 1-bit ADC Sampling, LEDs andPush buttons
- One PWM compare interface used with Timer2 to generate PWM testing pulse

A.3.2 Software Architecture

The software is designed around a real-time, embedded communication software architecture. Figure A-11 shows the ***main()*** function flow. There is a dead-loop (while(1)) after initializing microcontroller and RFIC front-end. The LPMF and Bit-Sync techniques are implemented inside of this dead-loop. A closer look at subsampling, Low Pass Matched Filter and Bit-sync software implementations will further be discussed.

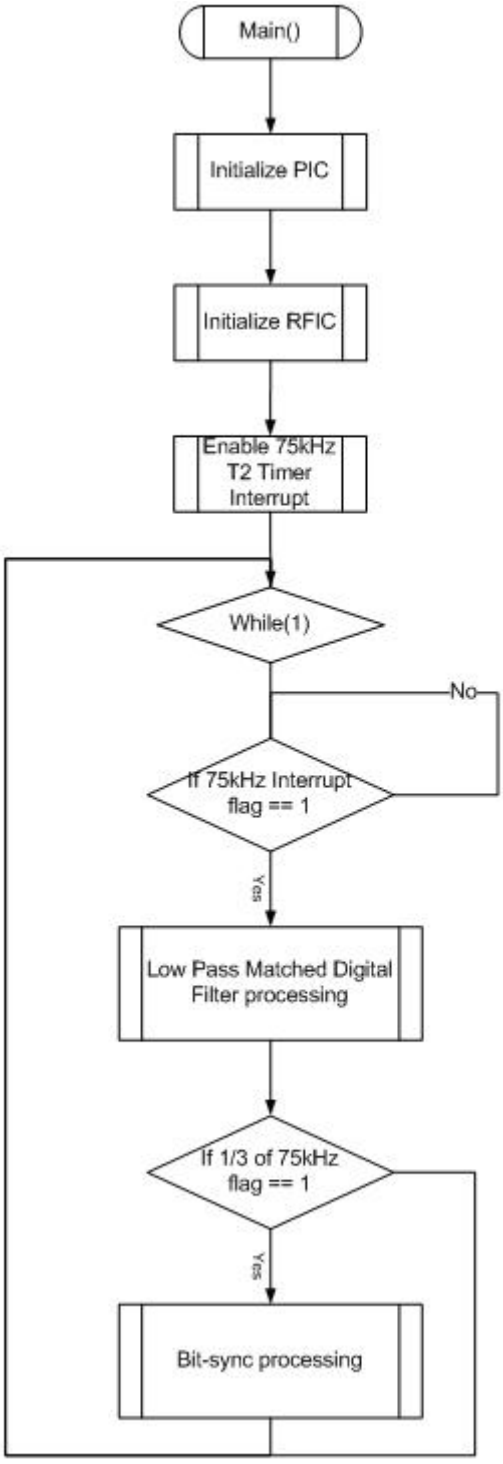


Figure 5-11 FSK Demodulation DSP software flow

The 75 kHz subsampling is implemented by using Timer2's interrupt on the microcontroller. Timer2 is enabled and its PR2 register, which is the timer period register, is set to 127 to generate 127* F_{CY} =13.2us = 75 kHz interrupt. Every 13.2us, the Timer2 Interrupt

Service Router (ISR) will be entered periodic. Inside Timer2 ISR, a 1-bit ADC sampling clock is sent to RFIC and a 1-bit ADC sample of IF signal is read back. Since zero crossings are detected, one history sample is stored. Two variables are used to do this, one is called _curSample _prevSample, which stores the current and previous samples respectively. Before a return from ISR, a 75 kHz sampling flag, _t2_int, is set to notify main() function to enter the low pass filter procedure and to clear the the Timer2 interrupt flag. Figure A-12 shows Timer2 ISR procedure flow.

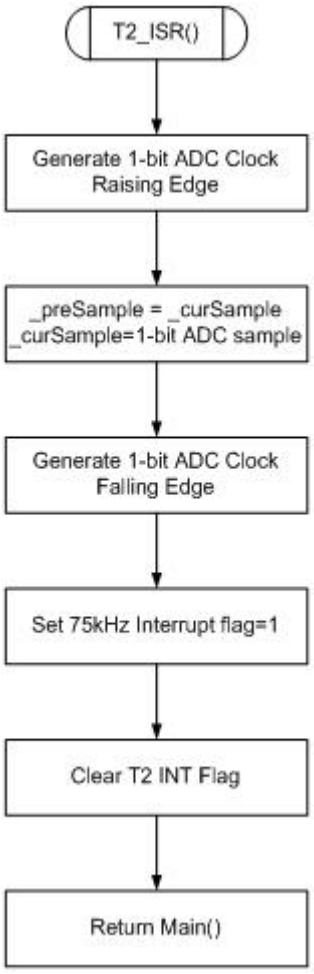


Figure 5-12 Timer2 ISR

The Low pass matched filter model is implemented inside of the main() function infinite loop. While in the infinite loop, the program is checking if a 1-bit ADC output has been sampled by checking the _t2_int variable. If it was, then it sets it to one then enters the LPMF module. Otherwise, it stays in the while dead loop. Figure A-13 shows the flow of LPMF module. The LPMF module has two routes to process the existence of zero crossings. Since the 1-bit ADC

outputs either a 0 or a 1, a _curSample of 1 and _prevSample of 0 indicates that a zero crossing has occurred. The sample is mapped to $+2^{14}$ and -2^{14} for a zero crossing and no zero crossing respectively. The current output of LPMF is denoted by y(n), which can be calculated via equation (4-9) $y(n) = b_2^{'}x(n) + b_1^{'}x(n-1) + b_0^{'}x(n-2) - a_1^{'}y(n-1) - a_2^{'}y(n-2)$. Because the IIR filter's coefficients are real numbers, floating point operations are required to solve this equation (4-9).

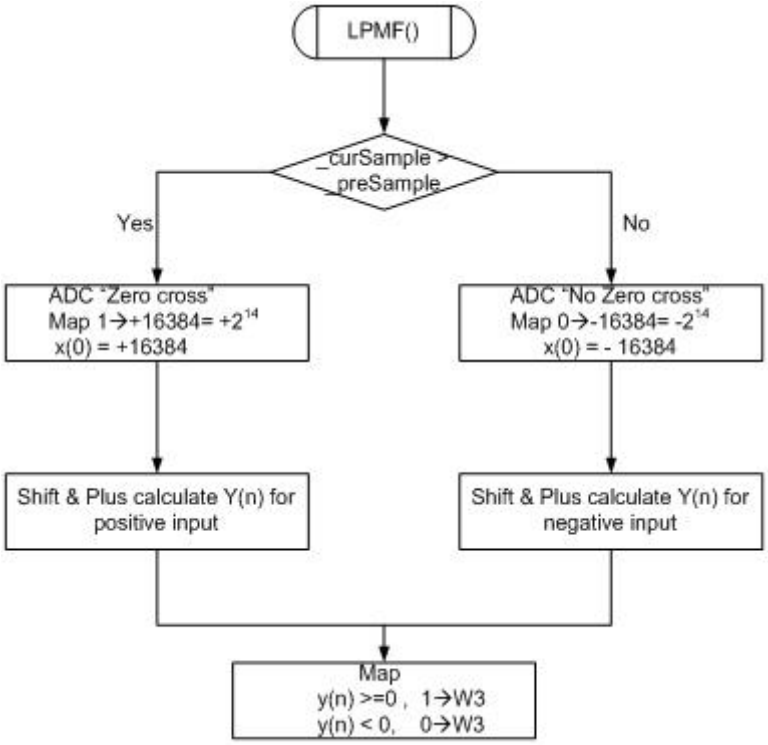


Figure 5-13 LPMF procedure

The real time system floating point operation requires at least two conditions: one is enough data width path and enough MIPS capability. It is easy to implement this equation by any program language if those two conditions are met. This system utilizes real-time DSP software on a microcontroller that offers a 16bit wide data path and operates on a 19.6 MHz clock.. It is difficult to describe a precision floating point data using 16-bit data wide path since the shortest float point standard of IEEE-754 is 32bits. The computational time complexity or time consumed by float multiplication on 19.6MHz clock speed will not be fast enough to match real-time requirement. Any integer number can be expressed as a sum of $\pm 2^p$ serial values where p is a positive integer. Any number smaller than 1 can be expressed as a sum of $\pm 2^n$ serial values

where n is a negative integer. For instance,$13=2^3 + 2^2 + 2^1$, $1.419=2^0 + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-8} + 2^{-10}$… The more components added, the result will be more precise. Thus a multiply operation could be expressed as $13*1.419= 13*(2^0 + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-8} + 2^{-10})$.

Therefore if any number multiplied by 2^p will be equal to this number left shifted by p bits, given that p is positive. If p is negative, any number multiplied with 2^pequal to this number right shifted by p bits. The Table A-2 shows right shift and left shift bits equivalent factors.

Table A-2 Bit shift and product factor

Right shift bits	Product factors	Right shift bits	Product factors	Left shift bits	Product factors	Left shift bits	Product factors
2^1	2	2^6	64	2^{-1}	0.5	2^{-6}	0.015625
2^2	4	2^7	128	2^{-2}	0.25	2^{-7}	0.007813
2^3	8	2^8	256	2^{-3}	0.125	2^{-8}	0.003906
2^4	16	2^9	512	2^{-4}	0.0625	2^{-9}	0.001953
2^5	32	2^{10}	1024	2^{-5}	0.03125	2^{-10}	0.000977

In this design, the 2 pole LPF digital filter coefficient are shown in Table 4-5,

a0=1	a1= -1.419	a2=0.5533
b0=0.0336	b1=0.0671	b2=0.0336

It can be re-wrote by using shift and product factor format in Table A-3.

Table A-3 LPF coefficients 2^n expression

Coefficient (float)	Coefficient (integer)
a0=1	2^0
a1=-1.419	$-(2^0 + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-8} + 2^{-10})$
a2=0.5533	$2^{-1} + 2^{-5} + 2^{-6} + 2^{-8} + 2^{-9} + 2^{-11}$
b0=0.0336	$2^{-5} + 2^{-9} + 2^{-12}$
b1=0.0671	$2^{-4} + 2^{-8} + 2^{-11}$
b2=0.0336	$2^{-5} + 2^{-9} + 2^{-12}$

Here is an example to calculate the filter transfer function equation (4-9) :

$$y(n) = b_2^{'}x(n) + b_1^{'}x(n-1) + b_0^{'}x(n-2) - a_1^{'}y(n-1) - a_2^{'}y(n-2)$$

Where x(i) and y(i) are integer number between -2^{14} and $+2^{14}$. The example assumes the 1st sample, $+2^{14}$, coming into LPF:

Since n=2, x(0) = x(1) = 0 , x(2) = $+2^{14}$=16384; y(0) = y(1) = y(2) =0.

Then the calculation is rewritten using shift and plus operation into:

$$y(n) = 16384 \times (2^{-5} + 2^{-9} + 2^{-12}) + 0 \times (2^{-4} + 2^{-8} + 2^{-11}) + 0 \times (2^{-5} + 2^{-9} + 2^{-12}) \\ - 0 \times (2^{0} + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-8} + 2^{-10}) - 0 \times (2^{-1} + 2^{-5} + 2^{-6} + 2^{-8} + 2^{-9} + 2^{-11}) \quad \text{Equ(4-11)}$$

The following assemble code in Figure A-14 are implemented the sum of coefficients ***b*** products parts of equation (4-11). Three components result are stored in ***w3***,***w4***,***w5*** register separately.

```
;Calculate X shifts
        mov _x1, w4                                     ; w4=x1
        mov  w4, _x2                                    ;  x2=x1
        mov _x2, w5                                     ;  w5=x2
        mov _x0, w3                                     ;  w3=x0
        mov  w3, _x1                                    ;  x1=x0,
        mov #0x4000, w3
        mov w3, _x0                                     ;  x0=16384

        ;( (_x0>>b0[0]) + (_x0>>b0[1]) + (_x0>>b0[2]) )
        asr w3, #5, w6                                  ;w6 = _x0>>5
        asr w3, #9, w7                                  ;w7 = _x0>>9
        asr w3, #12, w8                                 ;w8 = _x0>>12

        add w6, w7, w3                                  ;w3 = w6+w7
        add w3, w8, w3                                  ;w3 = w8+w3

        ;( (_x1>>b1[0]) + (_x1>>b1[1]) + (_x1>>b1[2]) )
        asr w4, #4, w6                                  ;w6 = _x1>>4
        asr w4, #8, w7                                  ;w7 = _x1>>8
        asr w4, #11, w8                                 ;w8 = _x1>>11

        add w6, w7, w4                                  ;w4 = w6+w7
        add w4, w8, w4                                  ;w4 = w8+w4

        ;( (_x2>>b2[0]) + (_x2>>b2[1]) + (_x2>>b2[2]) )
        asr w5, #5, w6                                  ;w6 = _x1>>5
        asr w5, #9, w7                                  ;w7 = _x1>>9
        asr w5, #12,w8                                  ;w8 = _x1>>12

        add w6, w7, w5                                  ;w5 = w6+w7
        add w5, w8, w5                                  ;w5 = w8+w5
```

Figure 5-14 LPF calculation coefficient *b* products

The Figure A-154-29 shows LPF implementation of sum of the coefficient a products and the two parts result are stored in ***w6*** and ***w7*** registers.

```
;Calculate Y shifts
        ;-(-(_y1>>a1[0]) - (_y1>>a1[1]) - (_y1>>a1[2]) - (_y1>>a1[3]) -
          (_y1>>a1[4]) - (_y1>>a1[5]) - (_y1>>a1[6]))
        ;asr w3, #0, wx                          ;shift 0
        mov _y1, w6
        asr w6, #2, w7                          ;w7 = _y1>>2
        asr w6, #3, w8                          ;w8 = _y1>>3
        asr w6, #5, w9                          ;w9 = _y1>>5
        asr w6, #7, w10                         ;w10= _y1>>7
        asr w6, #8, w11                         ;w11= _y1>>8
        asr w6, #10, w12                        ;w12= _y1>>10

        add w6, w7, w6                          ;w12= _y1>>0 + w7
        add w6, w8, w6                          ;w6 = w6+w8
        add w6, w9, w6                          ;w6 = w6+w9
        add w6, w10,w6                          ;w6 = w6+w10
        add w6, w11,w6                          ;w6 = w6+w11
        add w6, w12,w6                          ;w6 = w6+w12

        ;-( (_y2>>a2[0]) + (_y2>>a2[1]) + (_y2>>a2[2]) + (y2>>a2[3]) +
          (_y2>>a2[4]) + (_y2>>a2[5]) )
        mov _y2, w7
        asr w7, #1, w8                          ;w8 = _y1>>2
        asr w7, #5, w9                          ;w9 = _y1>>3
        asr w7, #6, w10                         ;w10= _y1>>5
        asr w7, #8, w11                         ;w11= _y1>>7
        asr w7, #9, w12                         ;w12= _y1>>8
        asr w7, #11,w13                         ;w13= _y1>>11

        add w8, w9, w7                          ;w7 = w8+w9
        add w7, w10,w7                          ;w7 = w7+w10
        add w7, w11,w7                          ;w7 = w7+w11
        add w7, w12,w7                          ;w7 = w7+w12
        add w7, w13,w7                          ;w7 = w7+w13
```

Figure 5-15 LPF calculation coefficients a products

Then, it needs to calculate the sum of all products as well as update y(i)s since x(i) was updated at the beginning of "Calculated X shifts" part. Figure A-16 shows this.

```
        ; lpf_out = w3+w4+w5-(-w6)-w7
        add w3, w4, w3                          ;w3 = w3+w4
        add w3, w5, w3                          ;w3 = w3+w5
        add w3, w6, w3                          ;w3 = w3+w6
        sub w3, w7, w0                          ;w0 = w3-w7

        ; update y0,y1,y2
        mov w0, _y0                             ;y0=lpf_out
        mov _y1, w4
        mov w4, _y2                             ;y2=y1
        mov w0, _y1                             ;y1=y0
```

Figure 5-16 Calculated X shifts

A.3.3 Test Result

In this section, the DSP software test results are discussed. The test environment shows in Figure A-17. A RIGOL DG 2021A function/Arbitrary waveform generator is used to generate the PHY layer frame including preamble, SOF and PHY payload. The generated bit stream feed into a HP 8648 Signal Generator. This Signal generator is doing FSK modulation as well as modulating the signal to 433.92MHz. The Agilent 54622D Oscilloscope is used to probe the output signal from circuit board. The HP ESA-L150000A Spectrum Analyzer is used to check the frequency spectrum.

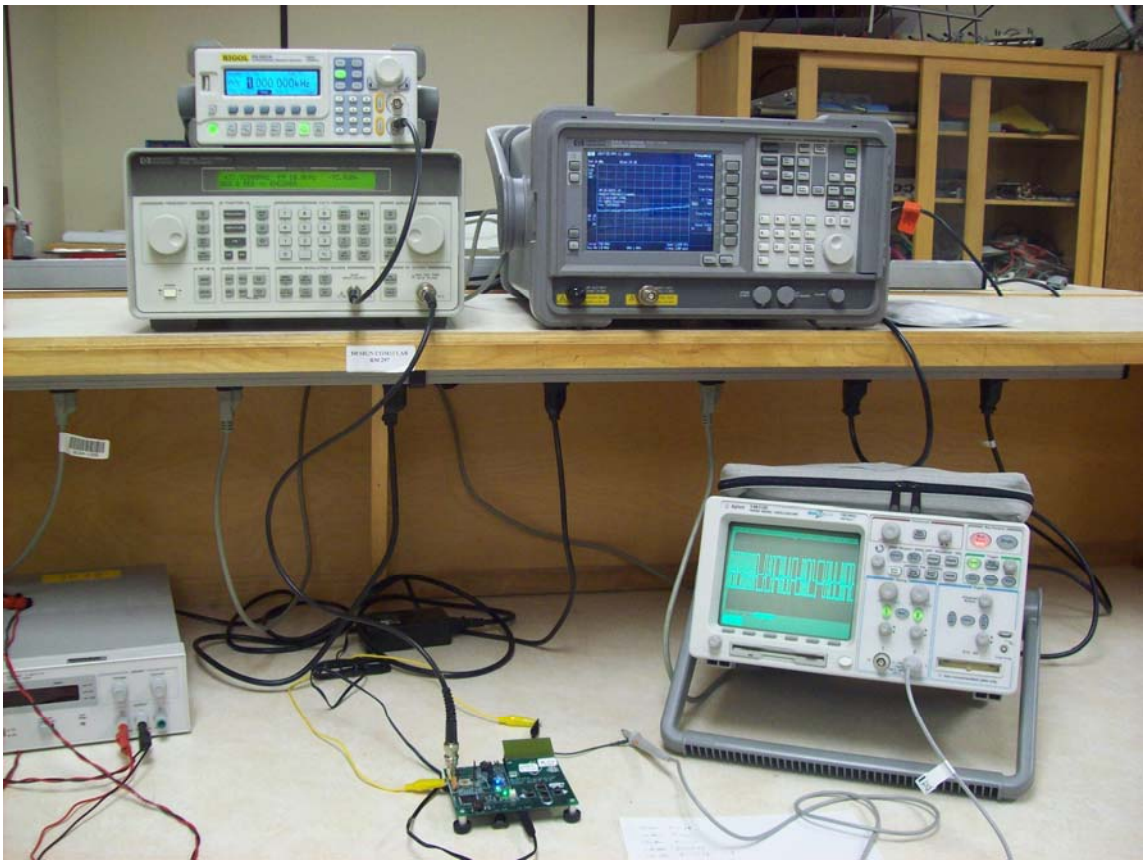


Figure 5-17 PHY Layer DSP test enviroment

1. One shot FSK demodulation test

First, we are designing 10kHz broad narrow band FSK modulation. So using the spectrum analyzer could check the modulation bandwidth. Figure A-18 shows the spectrum of FSK signal.

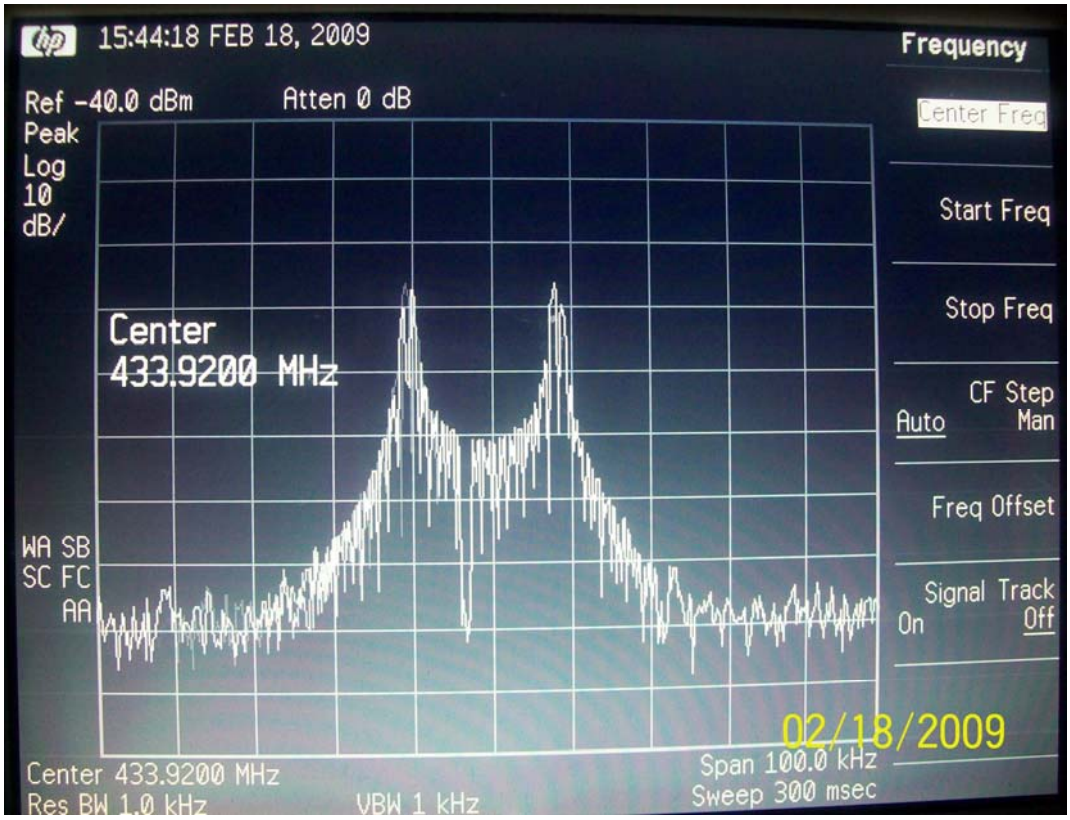


Figure 5-18 10kHz Bandwith FSK modulation

Second, the one-shot FSK demodulation method and digital filter were tested by generating PWM pulse. The one-shot FSK demodulation concept was shown in the Figure A-3. We would like to see if the pulse will be generated when a “zero-cross” is detected. Figure A-19 shows the PWM (Pulse Width Modulation) density corresponds with the transmitted bit stream screen capture from oscilloscope. The channel 2 signal is PWM pulse, which is generated by setting a microcontroller PWM timer. When a “zero-cross” is detected, a PWM pulse is

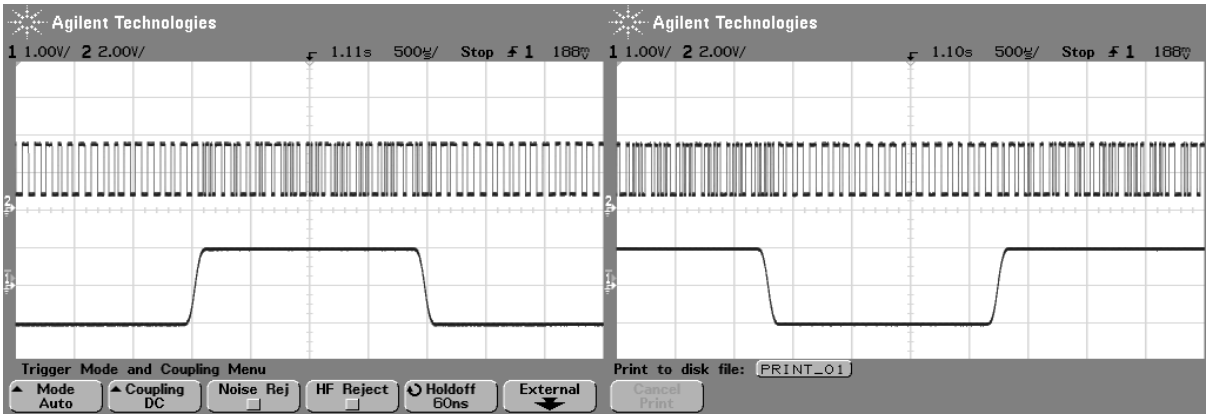


Figure 5-19 "Zero-cross" detection

generated and the pulse duty cycle is set depends on the 1bit ADC output value (1 or 0). In this case, when 1bit ADC output 1, the PWM pulse duty cycle is 100%. Otherwise, the duty cycle is 10%. The channel 1 signal is transmitted data. From Figure 4-33, the pulse density is distinctly higher when transmitted signal is 1.

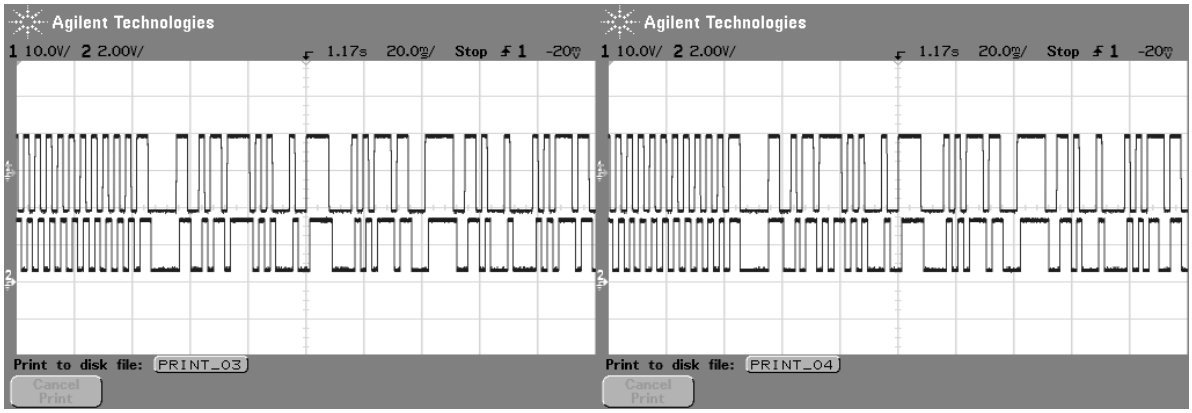
2. bit-sync test

The bit-sync is expected to generate a clock to lock the data sequence so that the raise edge of the clock locates at the right center of the bit period. Figure A-20 shows the test screen capture of bit sequence and the locked clock.

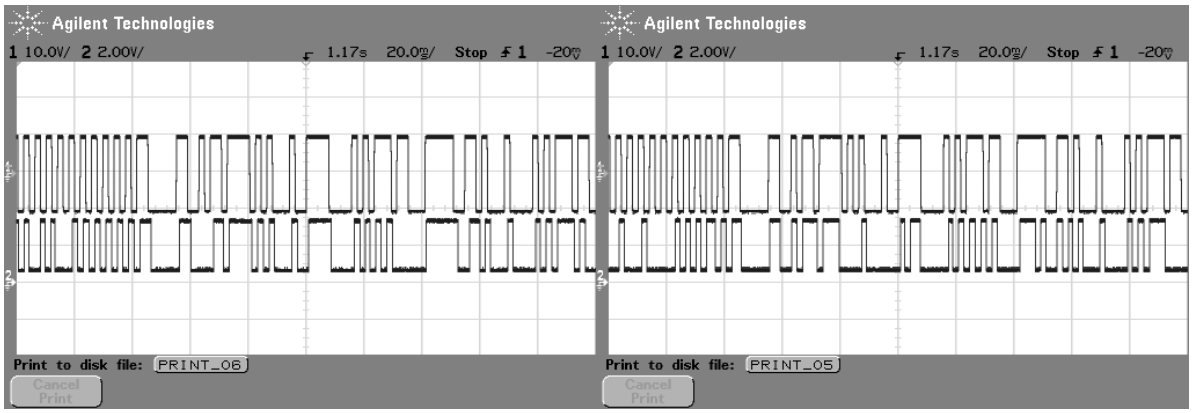
Figure 5-20 512bps Bit-sync Test restults

3. Data recovered test

The final goal is to recover the transmitted data. To test this, hook both the created arbitrary bits sequence and recovered data output pin to oscilloscope. The Figure A-21 is screen captures. The input signal power are -80dbm, -90dbm, -93dbm and -100dbm.



(a) -80dbm (b) -90dbm



(c) -93dbm (d) -100dbm

Figure 5-21 512bps Bit-sync Test restults

A.4 EHR DSP Code

A.4.1 ADC, LPF, and Bit-Sync File

```
;*************************************************************************
;  FileName: main.s                                                      *
;  Processor: PIC24HJ256 / External clock 19.2MHz --> Fcyl = 19.2/2 = 9.6MHz  *
;  Complier:  MPLAB C30 / ASM30                                          *
;  Linker:    MPLAB LINK30                                               *
;  Description : This is the file including the core function of ADC Sampling,*
;                            LPF and the Bit-sync in assembly language.     *
;  Author                                        Date           Version    *
;  Xiaohu Zhang                                  10/05/2008     v1.0       *
;*************************************************************************


;*********************************************************************
; Function : _main
;
; PreCondition: Implement 1-bit Sync;
;                  Doing 25kHz sample for 75kHz;
;                  To judge if this BIT is 0-cross
;                  Energy Harvesting Radio (EHR);
; Overview: This is the entrance of the Energy Harvesting Radio Physical
;                      Layer DSP implementation. The main function of this file
;                      includes 1bit ADC 75kHz subsampling, Zero-cross detector,
;                      1kHz two pole Low Pass Filter, 3:1 data decimation, and
;           Bit sync.
;*********************************************************************
.global _main
_main:

       call _main_c                        ;call main_c()

_IF_TIMER2_INT_:
       btss _t2_int,#0                     ; check the 75kHz Timer2 Interrupt flag
       goto _IF_TIMER2_INT_                ; goto process Timer2 interrupt requirement

       mov _curSample, w1                  ; Update sampling bit
       mov _prevSample,w2
       cp   w1,w2                          ; w1-w2 if curSample-prevSample=1 means
curSample=1&prevSample=0
       bra  LEU, ADC_NON_CROSS_            ; w1<=w2 unsigned less than or equal,jump to
_ADC_OUT_O_

;ADC 75kHz sample processing & 1kHz LPF
;Two bratch : one is ADC_0_CROSS_
;             one is ADC_NON_CROSS_
ADC_0_CROSS_:
       ;bset PORTE,#0                              ; used to test the input of LPF
;Calculate X shifts
       mov _x1, w4                                 ; w4=x1
       mov  w4, _x2                                ; x2=x1
       mov _x2, w5                                 ; w5=x2
       mov _x0, w3                                 ; w3=x0
       mov  w3, _x1                                ; x1=x0,
       mov #0x4000, w3
       mov w3, _x0                                 ; x0=16384

       ;( (_x0>>b0[0]) + (_x0>>b0[1]) + (_x0>>b0[2]) )
       asr w3, #5, w6                              ;w6 = _x0>>5
       asr w3, #9, w7                              ;w7 = _x0>>9
       asr w3, #12, w8                             ;w8 = _x0>>12

       add w6, w7, w3                              ;w3 = w6+w7
       add w3, w8, w3                              ;w3 = w8+w3
```

```
        ;( (_x1>>b1[0]) + (_x1>>b1[1]) + (_x1>>b1[2]) )
;       mov _x1, w4
        asr w4, #4, w6                          ;w6 = _x1>>4
        asr w4, #8, w7                          ;w7 = _x1>>8
        asr w4, #11, w8                         ;w8 = _x1>>11

        add w6, w7, w4                          ;w4 = w6+w7
        add w4, w8, w4                          ;w4 = w8+w4

        ;( (_x2>>b2[0]) + (_x2>>b2[1]) + (_x2>>b2[2]) )
        asr w5, #5, w6                          ;w6 = _x2>>5
        asr w5, #9, w7                          ;w7 = _x2>>9
        asr w5, #12,w8                          ;w8 = _x2>>12

        add w6, w7, w5                          ;w5 = w6+w7
        add w5, w8, w5                          ;w5 = w8+w5

;Calculate Y shifts
        ;-(-(_y1>>a1[0]) - (_y1>>a1[1]) - (_y1>>a1[2]) - (_y1>>a1[3]) - (_y1>>a1[4]) -
(_y1>>a1[5]) - (_y1>>a1[6]))
        ;asr w3, #0, wx                         ;shift 0
        mov _y1, w6
        asr w6, #2, w7                          ;w7 = _y1>>2
        asr w6, #3, w8                          ;w8 = _y1>>3
        asr w6, #5, w9                          ;w9 = _y1>>5
        asr w6, #7, w10                         ;w10= _y1>>7
        asr w6, #8, w11                         ;w11= _y1>>8
        asr w6, #10, w12                        ;w12= _y1>>10

        add w6, w7, w6                          ;w12= _y1>>0 + w7
        add w6, w8, w6                          ;w6 = w6+w8
        add w6, w9, w6                          ;w6 = w6+w9
        add w6, w10,w6                          ;w6 = w6+w10
        add w6, w11,w6                          ;w6 = w6+w11
        add w6, w12,w6                          ;w6 = w6+w12

        ;-( (_y2>>a2[0]) + (_y2>>a2[1]) + (_y2>>a2[2]) + (y2>>a2[3]) + (_y2>>a2[4]) +
(_y2>>a2[5]) )
        mov _y2, w7
        asr w7, #1, w8                          ;w8 = _y2>>2
        asr w7, #5, w9                          ;w9 = _y2>>3
        asr w7, #6, w10                         ;w10= _y2>>5
        asr w7, #8, w11                         ;w11= _y2>>7
        asr w7, #9, w12                         ;w12= _y2>>8
        asr w7, #11,w13                         ;w13= _y2>>11

        add w8, w9, w7                          ;w7 = w8+w9
        add w7, w10,w7                          ;w7 = w7+w10
        add w7, w11,w7                          ;w7 = w7+w11
        add w7, w12,w7                          ;w7 = w7+w12
        add w7, w13,w7                          ;w7 = w7+w13

        ; lpf_out = w3+w4+w5-(-w6)-w7
        add w3, w4, w3                          ;w3 = w3+w4
        add w3, w5, w3                          ;w3 = w3+w5
        add w3, w6, w3                          ;w3 = w3+w6
        sub w3, w7, w0                          ;w0 = w3-w7           Using w0 to return
value

        ; update y0,y1,y2
        mov w0, _y0                             ;y0=lpf_out
        mov _y1, w4
        mov w4, _y2                             ;y2=y1
        mov w0, _y1                             ;y1=y0
        ;goto MAP_LPF_OUT_
        ;goto BIT_SYNC_
goto DATA_BIT_1_



ADC_NON_CROSS_:
```

```
        ;bclr PORTE,#0                          ; used to test the input of LPF
        ;Calculate X shifts
        mov _x1, w4                             ; w4=x1
        mov  w4, _x2                            ; x2=x1
        mov _x2, w5                             ; w5=x2
        mov _x0, w3                             ; w3=x0
        mov  w3, _x1                            ; x1=x0,
        mov #0xc000, w3
        mov w3, _x0                             ; x0=-16384

        ;( (_x0>>b0[0]) + (_x0>>b0[1]) + (_x0>>b0[2]) )
        asr w3, #5, w6                          ;w6 = _x0>>5
        asr w3, #9, w7                          ;w7 = _x0>>9
        asr w3, #12, w8                         ;w8 = _x0>>12

        add w6, w7, w3                          ;w3 = w6+w7
        add w3, w8, w3                          ;w3 = w8+w3

        ;( (_x1>>b1[0]) + (_x1>>b1[1]) + (_x1>>b1[2]) )
;       mov _x1, w4
        asr w4, #4, w6                          ;w6 = _x1>>4
        asr w4, #8, w7                          ;w7 = _x1>>8
        asr w4, #11, w8                         ;w8 = _x1>>11

        add w6, w7, w4                          ;w4 = w6+w7
        add w4, w8, w4                          ;w4 = w8+w4

        ;( (_x2>>b2[0]) + (_x2>>b2[1]) + (_x2>>b2[2]) )
        asr w5, #5, w6                          ;w6 = _x2>>5
        asr w5, #9, w7                          ;w7 = _x2>>9
        asr w5, #12, w8                         ;w8 = _x2>>12

        add w6, w7, w5                          ;w5 = w6+w7
        add w5, w8, w5                          ;w5 = w8+w5

;Calculate Y shifts
        ;-(-(_y1>>a1[0]) - (_y1>>a1[1]) - (_y1>>a1[2]) - (_y1>>a1[3]) - (_y1>>a1[4]) -
(_y1>>a1[5]) - (_y1>>a1[6]))
        ;asr w3, #0, wx                         ;shift 0
        mov _y1, w6
        asr w6, #2, w7                          ;w7 = _y1>>2
        asr w6, #3, w8                          ;w8 = _y1>>3
        asr w6, #5, w9                          ;w9 = _y1>>5
        asr w6, #7, w10                         ;w10= _y1>>7
        asr w6, #8, w11                         ;w11= _y1>>8
        asr w6, #10, w12                        ;w12= _y1>>10

        add w6, w7, w6                          ;w12= _y1>>0 + w7
        add w6, w8, w6                          ;w6 = w6+w8
        add w6, w9, w6                          ;w6 = w6+w9
        add w6, w10,w6                          ;w6 = w6+w10
        add w6, w11,w6                          ;w6 = w6+w11
        add w6, w12,w6                          ;w6 = w6+w12

        ;-( (_y2>>a2[0]) + (_y2>>a2[1]) + (_y2>>a2[2]) + (y2>>a2[3]) + (_y2>>a2[4]) +
(_y2>>a2[5]) )
        mov _y2, w7
        asr w7, #1, w8                          ;w8 = _y2>>2
        asr w7, #5, w9                          ;w9 = _y2>>3
        asr w7, #6, w10                         ;w10= _y2>>5
        asr w7, #8, w11                         ;w11= _y2>>7
        asr w7, #9, w12                         ;w12= _y2>>8
        asr w7, #11, w13                        ;w13= _y2>>11

        add w8, w9, w7                          ;w7 = w8+w9
        add w7, w10,w7                          ;w7 = w7+w10
        add w7, w11,w7                          ;w7 = w7+w11
        add w7, w12,w7                          ;w7 = w7+w12
        add w7, w13,w7                          ;w7 = w7+w13
```

```
        ; lpf_out = w3+w4+w5-(-w6)-w7
        add w3, w4, w3                          ;w3 = w3+w4
        add w3, w5, w3                          ;w3 = w3+w5
        add w3, w6, w3                          ;w3 = w3+w6
        sub w3, w7, w0                          ;w0 = w3-w7 Using w0 to return value

        ; update y0,y1,y2
        mov w0, _y0                             ;y0=lpf_out=w0
        mov _y1, w4
        mov w4, _y2                             ;y2=y1
        mov w0, _y1                             ;y1=y0
        ;w0 still keeping lpf_out until BIT_25KHZ_SAMPLE_


DATA_BIT_1_:                                    ; output the data
; the number is used as a threshold caused by subsampling

        mov #0xEc78,w10    ;-5000 -1000 -2000 -3200 -3000 -2500 -1000 -2200 -1800 -1500 -1200  -
                           ;2000is good
        mov #0,w3
        cp w0,w10
        bra LT, BIT_SYNC_                       ; if(w0<w10) jump
        mov #1,w3

;1 Bit Sync code
;Doing 25kHz sample for 75kHz
;To judge if this BIT is 0-cross
BIT_SYNC_:
        dec _one_third                          ; one_third--
        bra NZ,RETURN_T2INT_                    ; if(one_third>0) jump
        mov #3, w1                              ; if(one_third==0) w1=3
        mov w1, _one_third                      ; reset one_third=3

UPDATE_BIT_:
        mov w3, _curBit                         ; update _curBit
        mov  _syncdBit,w4
        ;if transition happend
        cp w4,w3                                ; w3 = w4 - w3
        bra LEU,FSM_CHECK_                      ; w4<=w3 jump to FSM_CHECK_
        bset _transFlag,#0                      ; w4>w3 fall edge occured, set _transFlag

;Check current FSM if==23,24 or 25
FSM_CHECK_:
        ;Check current FSM sate
        mov _cur_FSM, w1
        cp w1,#24
        bra Z,SYNC_CLOCK_UP_                ; jump to generate 1BIT SYNC CLOCK

        ;mov #46,w14
        mov #38,w14
        cp w1,w14
        bra GEU,IF_TRANSITED_               ; (cur_FSM>=S22) jump to process TRANSITION DETECT
        goto FSM_TRANS_                     ; (cur_FSM<S22&&cur_FSM!=smp_FSM) FSM Transfer

IF_TRANSITED_:                              ; s23,s24,s25 and s25
        mov #49,w14
        cp w1,w14
        bra GEU, TRANS_2BITS_               ; (cur_FSM>=S25) jump
        cp0 _transFlag                      ; if(transFlag==0) for s23,s24,s25
        bra Z, FSM_TRANS_                   ; (transFlag==0) cur_FSM++
        goto RESET_FSM_              ; (transFlag==1) Transition happended then reset cur_FSM=0

; State machine transfer
FSM_TRANS_:
        inc _cur_FSM
        clr _transFlag                      ; clear the transflag if cur_FSM<47
        goto RETURN_T2INT_

; Recorve Data output
SYNC_CLOCK_UP_:
        mov _curBit,w3
```

```
        mov w3,_syncdBit
        inc _cur_FSM
        clr _transFlag                      ; clear the _transFlag
        bset PORTD,#0

        ;Update Start Of Frame
        SL _StartOfFrame                    ; left shift 1bit of Start Of Frame

        ;Recovered Data
        CP0 w3
        bra Z, DATA_BIT_0_
        bset PORTE,#3
        bset _StartOfFrame,#0               ; update lowest SOF bit=1
        goto RETURN_T2INT_
DATA_BIT_0_:
        bclr PORTE,#3
        bclr _StartOfFrame,#0               ; update lowest SOF bit=0

; Detect Start Of Frame
SOF_DETEC_:
        mov _StartOfFrame, w4               ; w4 = _StartOfFrame
        and #0x000f,w4                      ; w4 = _StartOfFrame & 0x000f
        xor w4,#0x000e,w5                   ; w4 = 0x00001110 xor _StartOfFrame[0:3]
        cp0 w5                              ; w5=1:ERROR_SOF; w5=0:OK_SOF.
        bra NZ, SOF_ERROR
        bset PORTE,#1                       ; Generate pulse to indicate Correct SOF

SOF_ERROR:
; End of Detect SOF
        goto RETURN_T2INT_

TRANS_2BITS_:                               ; there are two "0" or "1" transmitting without
transition between
;       cp0 _transFlag                      ; if(transFlag==1) jump to state0
;       bra NZ, RESET_FSM_
        mov #1,w1                   ; them. Update the state from 26(2 laters than 24) to 3
        mov w1,_cur_FSM
        goto SYNC_CLOCK_DOWN_               ; goto return
        ;goto SYNC_BIT_OUT1_

RESET_FSM_:
        clr _cur_FSM                        ; reset cur_FSM=0;
        clr _transFlag                      ; clear transFlag=0;

SYNC_CLOCK_DOWN_:
        bclr PORTD,#0                       ; RD0=0 to pull down BIT_SYNC_CLOCK down edge
        ;goto SAMPLING_CLK_LOW_             ; goto next sample loop

;Must make this before setting TMR6 in case PR6<2
;(Only RETURN_T2INT 2 instructions after TRM6 running)
RETURN_T2INT_:
        bclr PORTE,#1                       ; Clear pulse of SOF indication
        bclr _t2_int, #0                    ; clear _T2IF interrupt flag
        goto _IF_TIMER2_INT_

return  ;
```

A.4.2 PIC Initialization file

```
/*****************************************************************************
*  FileName : IniPIC.c
*  Processor: PIC24HJ256 / External clock 19.2MHz --> Fcyl = 19.2/2 = 9.6MHz
*  Complier:  MPLAB C30 / ASM30
*  Linker:    MPLAB LINK30
*  Description : This is the function file in which the C functions are included,
*                sunch as microprocessor initilization functions.
*
*  Author                                   Date                  Version
*  Xiaohu Zhang                             10/05/2008            v1.0
*****************************************************************************/
#include "LPF_H.h"
#include "lpf_buf.h"

//__CONFIG (GSS_OFF);

void main_c();
double LPF_1BIT_FLOAT(unsigned char in_bit);
inline signed short int LPF_1BIT_INT(signed short int in_16int);
//void LFP(double *a, double *b, unsigned char *buf, double buf_len, double *lpf_out);
void pic_init();
void timer_init();
void interrupts_init();
void activateRxMode(void);                                  //Set RFIC RX mode Copy from Joe's
code
void RxDemod(void);                                                 //Set RFIC FM Demodulation
mode Cope from Joe's code

inline void timer6_start(short int pr);
inline void timer6_stop();

typedef struct _BUF_{
       unsigned char bit0 :1;
    unsigned char bit1 :1;
       unsigned char bit2 :1;
       unsigned char bit3 :1;
       unsigned char bit4 :1;
       unsigned char bit5 :1;
       unsigned char bit6 :1;
       unsigned char bit7 :1;
}BUF;

/*LPF coefficient*/
double a[3]={1.0,   -1.4190, 0.5533};
double b[3]={0.0336, 0.0671, 0.0336};

/*Right Shifts & Plus bits Filter coefficients*/
signed int a0=0;
signed int a1[]={0,2,3,5,7,8,10};
signed int a2[]={1,5,6,8,9,11};
signed int b0[]={5,9,12};
signed int b1[]={4,8,11};
signed int b2[]={5,9,12};

/*LPF input parameter*/
volatile signed short int x0=0, x1=0, x2=0;
volatile signed short int y0=0, y1=0, y2=0;

short int tmr6_pr=120;
short int MY_PWM=128;
unsigned int asm_buf;
unsigned short int asm_lpf_out_int;
unsigned short int asm_lpf_out_lev;
short int asm_pr1;

/*Input Sample*/
volatile signed short int IN_SAMPLE=0;

/* 512bps Bit-sync State Machine*/
```

```
enum FSM{S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10,S11,S12,
          S13,S14,S15,S16,S17,S18,S19,S20,S21,S22,S23,S24,S25,S26,
          S27,S28,S29,S30,S31,S32,S33,S34,S35,S36,S37,S38,S39,S40,
          S41,S42,S43,S44,S45,S46,S47,S48,S49,S50};
enum FSM cur_FSM=S0;                        // Current State
enum FSM pre_FSM=S0;                        // Previous State
enum FSM smp_FSM=S12;                       // Sampling state
unsigned short int curBit=0;                // Current 25kHz bit sample
unsigned short int prevBit=0;               // Previous 25kHz bit sample
unsigned short int transFlag=0;             // Transition flag 0-No 1-Yes
unsigned short int syncdBit=0;              // Syncronized Output Bit
unsigned short int StartOfFrame=0;          //Start of Frame string
volatile unsigned short int one_third=3;    // 1/3 of 75KHZ sampling rate
volatile unsigned short int t2_int=0; //flag of Timer2 interrupt 1:INT happened; 0:NO INT.

/*****************************************************************************
*  Function : void main()
*  Input:
*  Return:
*  Description: This function finish PIC initialization.
*
*  Author                                   Date                  Version
*  Xiaohu Zhang                             10/05/2008            v1.0
*****************************************************************************/
void main_c()
{
       signed short int in_sint=0;
       signed short int lpf_out_int[sizeof(buf)/6];
    signed short int lpf_out_lev[sizeof(buf)/6];

       long int buf_len=sizeof(buf)/6;
       long int i=0;
       short int pr=0;
       short int pr1[sizeof(buf)/6];


       memset(lpf_out_int,0,sizeof(lpf_out_int));
       memset(lpf_out_lev,0,sizeof(lpf_out_int));
       memset(pr1,0,sizeof(pr1));
       pic_init();
       initRFICBits();

       _TRISE0 = 0;
       _TRISE1 = 0;
       _TRISE2 = 0;
       _TRISE3 = 0;
       _RE0 = 0;
       _RE1 = 0;                                    //Light LED
       _RE2 = 0;


       activateRxMode();
       RxDemod();
};


/*****************************************************************************************
*  Function : int pic_init(void)
*  Input:         NONE;
*  Return:        int -- sucessful / fail.
*
*  Description: Initialize PIC.
*
*
*  Author                                          Date                                  Version
*  Xiaohu Zhang                                    10/05/2008                            v1.0
*****************************************************************************************/
void pic_init()
{
       timer_init();
       gpio_init();
```

```
        initSPI1();
        interrupts_init();

        //If using PWM should open this block
        //Initialize OCM
/*
        OC1CONbits.OCM = 0b000;       // Disable OCM
        OC1RS= 0;
        OC1CONbits.OCTSEL = 0; // Select Timer2
        OC1R = 0;
        OC1CONbits.OCM = 0b110;       // PWM mode on OC1
*/

}
/*****************************************************************************
*  Function : int gpio_init(void)
*  Input:         NONE;
*  Return:        NONE;
*
*  Description: Initialize PIC GPIO.
*
*
*  Author                                          Date                              Version
*  Xiaohu Zhang                                    10/05/2008                        v1.0
*****************************************************************************/
void gpio_init(void)
{
        _TRISD0  = 0;        //Set as BIT_SYNC_CLOCK as Output
    _RD0 = 0;                  //Set low
        _TRISD6  = 0;        //Set RFIC_ADC_CLK_P as Output
        _TRISD7  = 0;        //Set RFIC_ADC_CLK_N as Output
        _TRISD9  = 1;        //Set RFIC_ADC_OUT_P as Input
        _TRISD10 = 1;        //Set RFIC_ADC_OUT_N as Input

        _TRISE0 = 0;           //Set LED as Output
        _TRISE1 = 0;           //Set LED as Output
        _TRISE2 = 0;           //Set LED as Output
        _TRISE3 = 0;           //Set LED as Output
}
/*****************************************************************************
*  Function : int timer_init(void)
*  Input:         NONE;
*  Return:        NONE;
*
*  Description: Initialize PIC TIMER.
*
*
*  Author                                          Date                              Version
*  Xiaohu Zhang                                    10/05/2008                        v1.0
*****************************************************************************/
void timer_init(void)
{
        /*Timer2/3 Configuration*/
        T2CON = 0B0000000000000000;
        _T2IP = 0x02;                 /*  _TON = 0;   // Stop timer
                                          _TSIDL = 1; // Stop in idle mode
                                          _TGATE = 0; // Gated time accumulation disabled
                                          _TCKPS = 0; // 1:1 prescale
                                          _T32   = 1; // 32-bit timer
                                          _TCS   = 0;       // Internal clock Fcy
                                          _T2IP  = 2; // Timer2 propority is 2 lower than T6
                                      */


        /*Timer6 Configure as 16 bits timer*/
        T6CON = 0B0000000000000000;
        _T6IP = 0x02;         // Set the priority of Timer6
        PR6 = 0x0;                     /* _TON = 0;   // Stop timer
                                          _TSIDL = 1; // Stop in idle mode
                                          _TGATE = 0; // Gated time accumulation disabled
                                          _TCKPS = 0; // 1:1 prescale
```

```
                                    _T32   = 1; // 32-bit timer
                                    _TCS   = 0;       // Internal clock Fcy
                                 */
        /*Timer7 Configure as 32 bits timer*/
        T7CON = 0B0000000000000000;
        PR7 = 0x0;                   /* _TON = 0;   // Stop timer
                                    _TSIDL = 1; // Stop in idle mode
                                    _TGATE = 0; // Gated time accumulation disabled
                                    _TCKPS = 0; // 1:1 prescale
                                    _T32   = 1; // 32-bit timer
                                    _TCS   = 0;       // Internal clock Fcy
                                 */

        /*Timer8 Configure as 16 bits timer*/
        T8CON = 0B0000000000000000;
        PR8 = 0x0;                  /* _TON = 0;   // Stop timer
                                    _TSIDL = 1; // Stop in idle mode
                                    _TGATE = 0; // Gated time accumulation disabled
                                    _TCKPS = 0; // 1:1 prescale
                                    _T32   = 0; // 16-bit timer
                                    _TCS   = 0; // Internal clock Fcy
                                 */

        /*Timer9 Configure as 16 bits timer*/
        T9CON = 0B0010000000000000;
        PR9 = 0x0;                   /* _TON = 0;   // Stop timer
                                    _TSIDL = 1; // Stop in idle mode
                                    _TGATE = 0; // Gated time accumulation disabled
                                    _TCKPS = 0; // 1:1 prescale
                                    _T32   = 0; // 16-bit timer
                                    _TCS   = 0;       // Internal clock Fcy
                                 */
}

/*****************************************************************************
*  Function : int interrupts_init(void)
*  Input:         NONE;
*  Return:        NONE;
*
*  Description: Initialize PIC.
*
*
*  Author                          Date                                Version
*  Xiaohu Zhang                    10/05/2008                          v1.0
*****************************************************************************/
void interrupts_init()
{
        /* Initialize & Configure Interrupts */
        SR     = 0x00;
        CORCON = 0x0c;       //causes the lcd to break
        INTCON1= 0x00;
        INTCON2= 0x1f;       // 0b0000000000011111 set fall-edge interrupts sensitive

        /*clear all interrupts flag*/
        IFS0 = 0;
        IFS1 = 0;
        IFS2 = 0;
        IFS3 = 0;
        IFS4 = 0;

        _T2IE = 1;                  //Enable Timer2
        _T3IE = 1;                  //Enable Timer3
        _T4IE = 1;                  //Enable Timer4
        _T5IE = 1;                  //Enable Timer5
        _T6IE = 1;                  //Enable Timer6
        _T7IE = 1;                  //Enable Timer7
        _T8IE = 1;                  //Enable Timer8
        _T9IE = 1;                  //Enable Timer9

        AD1PCFGL = 0xFFFF;     //Configure ADC1 ANx pins as digital I/O
        AD1PCFGH = 0xFFFF;     //Configure ADC1 ANx pins as digital I/O
```

```
        AD2PCFGL = 0xFFFF;     //Configure ADC2 ANx pins as digital I/O

        _CN2IE = 1;            //Enable CN2 interrupt (UP Button)
        _CN3IE = 1;            //Enable CN3 interrupt (DN Button)
        _CN4IE = 1;            //Enable CN4 interrupt (SEL Button)
        _CN5IE = 1;            //Enable CN5 interrupt (LFT Button)
        _CN6IE = 1;            //Enable CN6 interrupt (RGT Button)

        _CNIF = 0;             //Clear Change Notification Interrupt Flag
        _CNIE = 1;             //Enable Change Notification Interrupts
        _CNIP = 5;             //Change Notification Interrupt Priority bits

        /* SPI1 Transfer Complete Interrupt Configuration */
        _SPI1IF = 0;           //Clear SPI1 Event Interrupt Flag Status bit
        _SPI1IE = 1;           //Enable SPI1 Transfer Complete Interrupt
        _SPI1IP = 7;           //SPI1 Event Interrupt Priority bits (Highest Priority)
}
```

A.4.3 ISR Interrupt Process file

```
;*****************************************************************************
;  FileName : isr_.asm
;  Processor: PIC24HJ256 / External clock 19.2MHz --> Fcyl = 19.2/2 = 9.6MHz
;  Complier:  MPLAB C30 / ASM30
;  Linker:    MPLAB LINK30
;  Description : This is the Interrupt Services Route function file.
;  Author                                        Date                              Version
;  Xiaohu Zhang                                  10/05/2008                        v1.0
;*****************************************************************************/
.include "p24hj256gp610.inc"

;Global variables

;.text

;Global function be called in C
.global _AsmReset
.global __T2Interrupt
.global __T6Interrupt

_AsmReset:
       clr _x0
       clr _x1
       clr _x2
       clr _y0
       clr _y1
       clr _y2

;*********************************************************************;
; Function: T2Interrupt
; Decription : Timer 2 ISR.                                           ;
;                                                                     ;
;*********************************************************************;

__T2Interrupt:

       push w0
       push w1

SAMPLING_CLK_HIGH_:
       mov #0xff3f,w0
       and PORTD,WREG                      ; w0 = PORTD & w0
       ior #0x80,w0
       mov w0, PORTD

; Update prevSample = curSample
       mov  _curSample, w1
       mov  w1,_prevSample


;Update current 75kHz sample
UPDATE_SAMPLE_:
       mov  #1, w1                         ; w1 = 1
       btss PORTD,#0x9                     ; if RD9==1 jump to ADC_OUT_1;
       mov  #0, w1                         ; (RD9==0) w1 = 0;
       mov  w1, _curSample                 ; update _curSample

;Must make this before setting TMR6 in case PR6<2
;(Only RETURN_T2INT 2 instructions after TRM6 running)
SAMPLING_CLK_LOW_:
       mov #0xff3f,w0
       and PORTD,WREG                      ; w0 = PORTD & w0
       ior #0x40,w0
       mov w0, PORTD

       pop w1
       pop w0

;Set the timer2 interrupt flag
```

```
       bset _t2_int,#0
;Clear the interrupt flag
       bclr IFS0,#7
RETFIE  ;return from ISR


;*********************************************************************;
; Function: _T6Interrupt                                             ;
; Description : Timer 6 ISR.                                         ;
;*********************************************************************;
_myT6Interrupt:
;      bclr PORTD, #0                      ; _RD0=0 Pull down PWM
       bclr LATD,#0
       clr PR6                             ; PR6=0 clear timer period register
       bclr T6CON, #15                     ; _TON=0 Stop Timer6
       bclr IFS2,  #15                     ; _T6IF=0 clear Timer6 interrupt flag
       return
;.end
```

Appedix B - K-State Energy Harvesting Demo Board Schematic

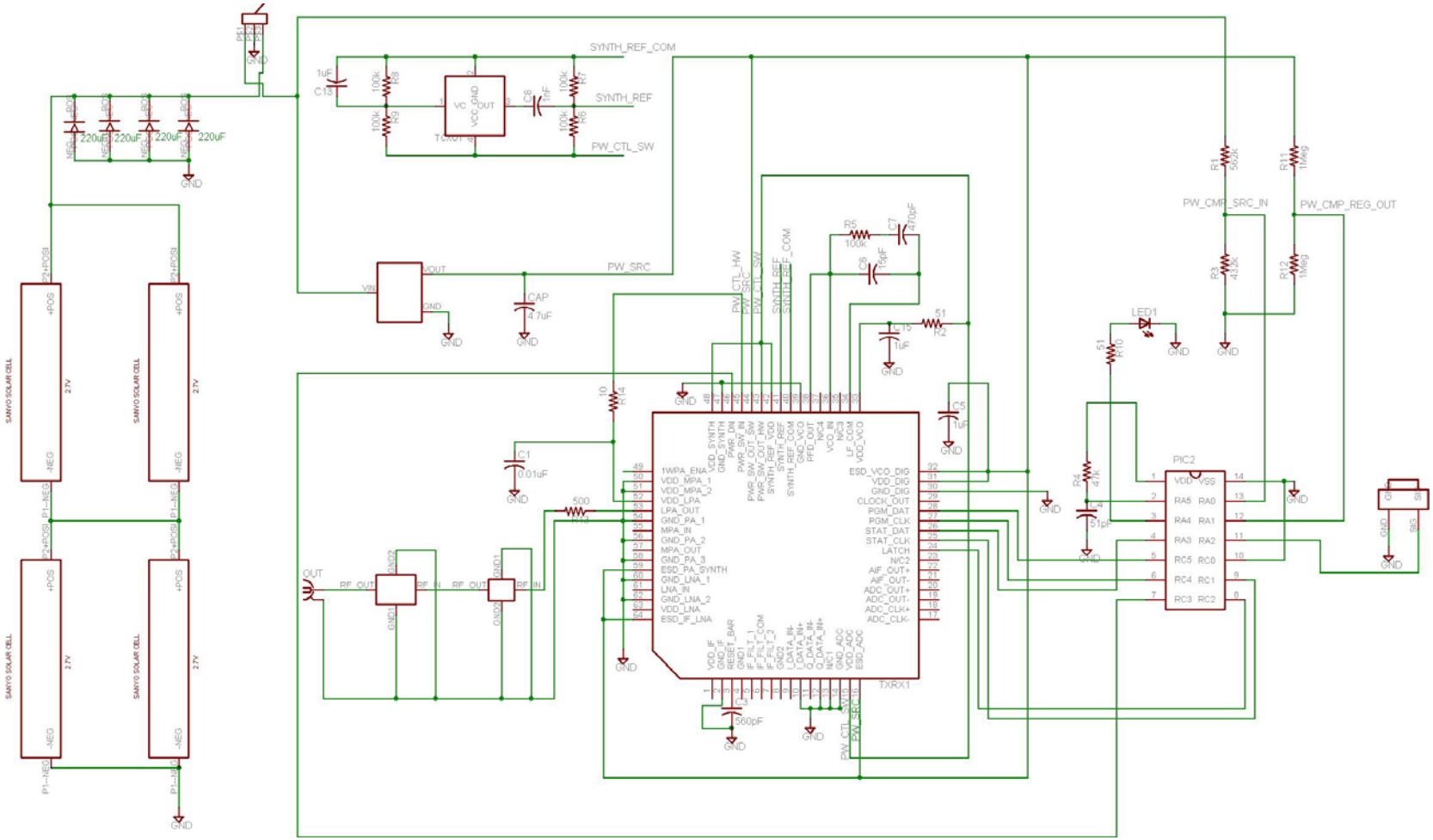


Figure 5-22 Schematic

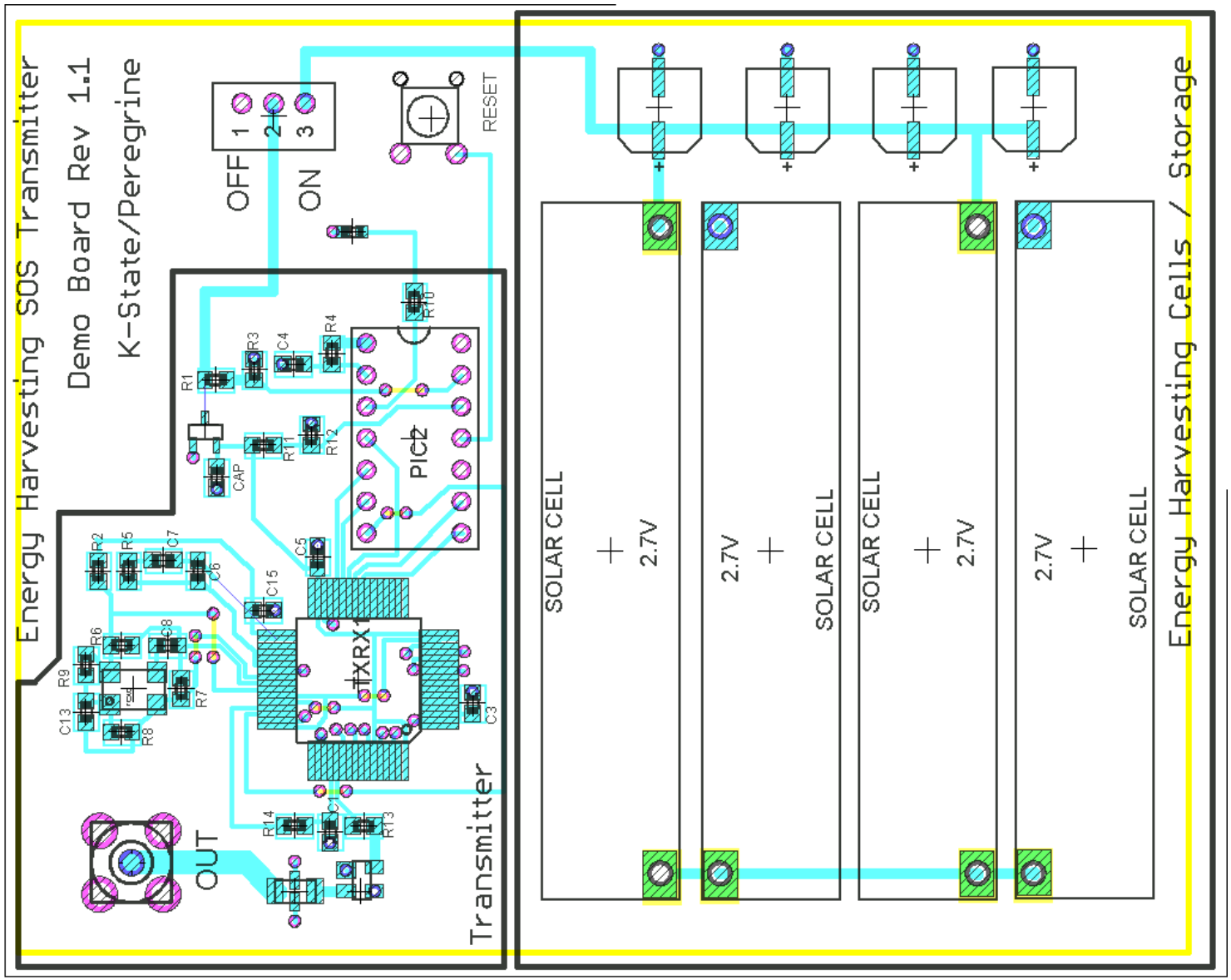


Figure 5-23 Layout

Appedix C - Frequency Synthesis Board Schematic and Layout

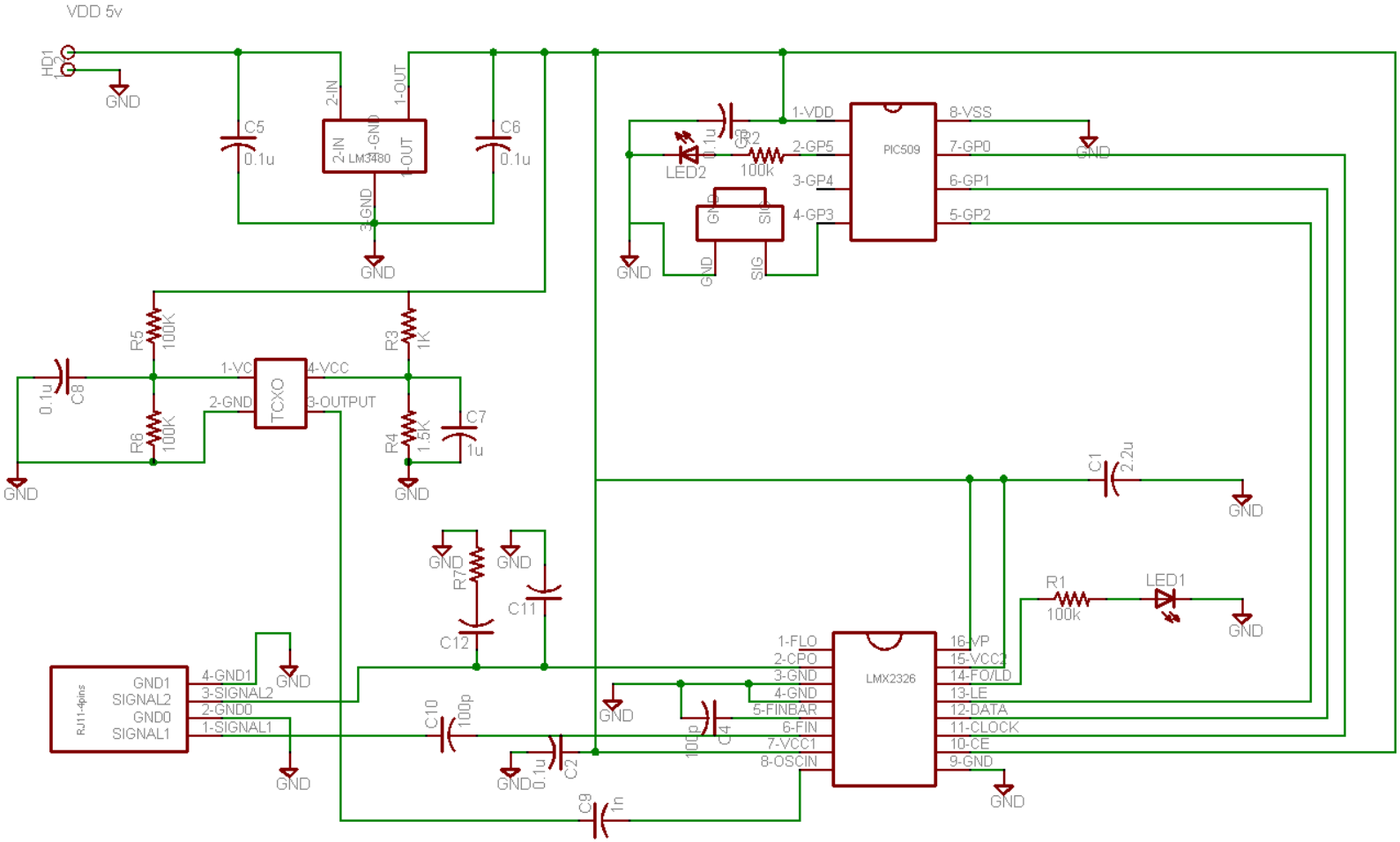


Figure 5-24 Schematic

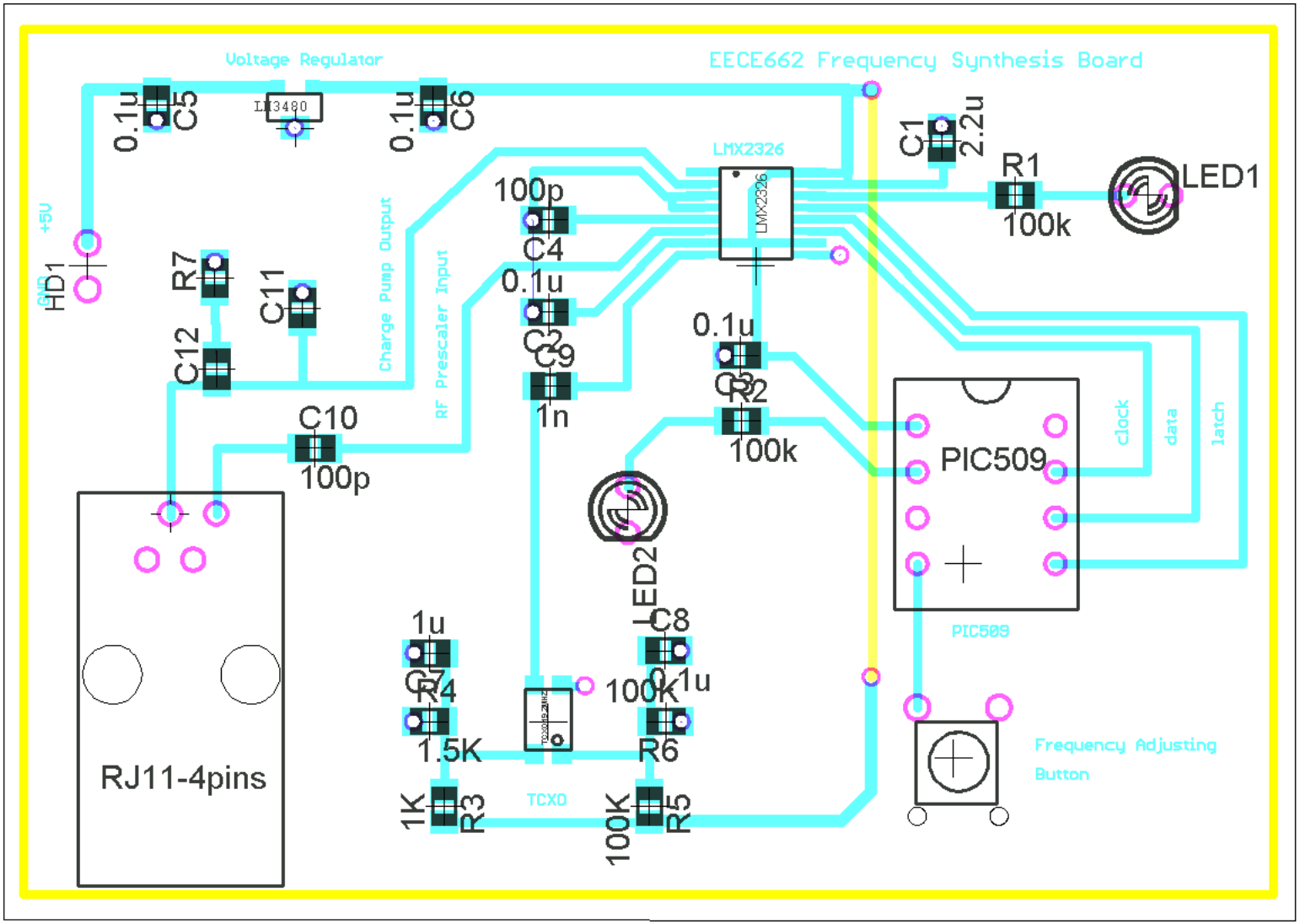


Figure 5-25 Layout

Appedix D - K-State EHR Demo Board Code for 4MHz CPU Clock

```
; Program peregine1#200uF10uF4M.asm
;
; Test program for Peregine 1# board.
; By  William B. Kuhn, Xiaohu Zhang
; Created 08/13/07.
;
;
; TEST PLAN 2. The RFIC working depends on CC0, CCpic charge and discharge.
;*********************
; CC0   = 200uF
; CC1   = N/A
; CCpic = 10uF
;*********************
;Design:
;Two charging capacitor are used here. One is CC0=200uF used to drive RFIC working for 0.012s at
least; another is CCpic=10uF used to drive Microchip PIC16f676 working at 4MHz.
;
;
; This program provides the 60 bits of programming needed to set
; the fab4 transceiver chip to transmitt mode with full gain and sleep mode
; enabled. To keep low power consuming, we use <1% duty cycle that means during 1 second, xx ms
are occupied by RFIC to transmitt data and the remain time, xxx ms, RFIC will enter sleep mode to
save power.
; The synthesizer is setup to transimitt a frequency close to CCSDS channel 1
; using integer-N mode.  The transmit freq is at 404.4 MHz.
; In the control field, LPAena is set to 1 to put the TR switch into transmit mode and Stby_bar
is set to 0 to allow RFIC slepping. See other programs for details on the programming bits.


; Meta data for the compiler
       list    p = 16f676
       radix   dec
       include <p16f676.inc>
       errorlevel  -302             ; suppress arg range msg from list file

; Configure the chip for Code protect off, Data Code protect off, Brownout detect off, MCLR
disabled,
; WDT disabled, Powerup timer enabled, and internal oscillator (with no use of output pin)

    __CONFIG   _CP_OFF & _CPD_OFF & _BODEN_OFF & _MCLRE_OFF & _WDT_ON & _PWRTE_ON &
_INTRC_OSC_NOCLKOUT

; '__CONFIG' directive is used to embed configuration word within .asm file.
; The lables following the directive are located in the respective .inc file.
; See data sheet for additional information on configuration word settings.

; Symbolic constants

REG_EN         equ 0   ; RC0 (pin 10) ; Voltage regulator enable signal

STATCLK        equ 1   ; RC1 (pin 9)  ; Serial programming input and status output register
control pins
LATCH          equ 2   ; RC2 (pin 8)
STATDAT        equ 3   ; RA3 (pin 4)
PGMCLK         equ 4   ; RC4 (pin 6)
PGMDAT         equ 5   ; RC5 (pin 5)

PW_CMP_H       equ 0   ; RA0 (pin13)  ; Solar cell voltage compare
PW_CMP_L       equ 1   ; RA1 (pin12)
PIC_RST        equ 2   ; RA2 (pin11)  ; PIC reset button
RFIC_PW_DN     equ 3   ; RC3 (pin7)   ; RFIC Power Down pin
LED            equ 4   ; RA4 (pin3)   ; LED
```

```
; Register assignments

        cblock 0x20
w_temp                      ; temporary used in (unimplemented) ISR at 0x004
status_temp                 ; ditto
pclath_temp                 ; temporary used in ISR
numbits                     ; # of bits to write to rfic
temp                        ; temporary
count                       ; counter used in delay function
stat_group                  ; which status grouping to read / display
                            ; 0 = temp, 1 = synthlock, 2 = rssi, 4 = spare
temperature                 ; variable to shift status bits into
synth_lock                  ; variable to shift status bits into
rssi                        ; variable to shift status bits into
pulse_cnt                   ; # of pulse send during wake up period, which is
                            ; determined by the frequency of modulation signal
reg_ctl                     ; control subfield
tmr1_int_flg                ; flag used to indicate whether timer interrupt occured
tmr1_int_cnt                ; # of timer1 interrupts #*0xffff
tmr1_H                      ; the left timing number high byte
tmr1_L                      ; the left timing number low byte
sleep_cnt                   ; # of WDT sleep period depends on the WDT presca
        endc


; *****************  Start of program *************************

; Setup reset vector to skip over interrupt subroutines, if any
        org     0x00
                goto    start

; An interrupt service routine template.  (Not used in this program)
        org 0x004             ; interrupt vector location
                ; SAVE current status register
                movwf   w_temp           ; save off current W register contents
                movf    STATUS,w         ; move status register into W register
                movwf   status_temp      ; save off contents of STATUS register
                movf    PCLATH,w         ; move pclath register into w register
                movwf   pclath_temp      ; save off contents of PCLATH register


                ; isr code can go here or be located as a call subroutine elsewhere
                clrf    STATUS
                btfsc   PIR1, TMR1IF     ; timer1 overflow interrupt
                call    TMR1_INT         ; yes

                btfsc   INTCON, INTF     ; if RA2 pin interrupt occure
                call    RA2_INT          ; yes


                ; RESTORED status register
                movf    pclath_temp,w     ; retrieve copy of PCLATH register
                movwf   PCLATH            ; restore pre-isr PCLATH register contents
                movf    status_temp,w     ; retrieve copy of STATUS register
                movwf   STATUS            ; restore pre-isr STATUS register contents
                swapf   w_temp,f
                swapf   w_temp,w          ; restore pre-isr W register contents
                retfie                    ; return from interrupt


; ************************************
; The main routine
; ************************************

start
                ;  Calibration of internal oscillator (from 16f676 template code)
                bsf     STATUS,RP0        ; set file register bank to 1
                call    0x3FF             ; retrieve factory calibration value
                movwf   OSCCAL            ; update register with factory cal value
                bcf     STATUS,RP0        ; set file register bank to 0
```

```
                ; initialize the PIC and its I/O ports
                call    initPIC

                bcf     PORTC, REG_EN         ; Disable voltage regulator wait for CC0 charging
                bsf    PORTC, RFIC_PW_DN      ; Power down RFIC

                clrwdt                        ; clear WDT

                bsf     PORTA, LED
                call    wait10ms
                bcf     PORTA, LED
                call    wait10ms
                bsf     PORTA, LED
                call    wait10ms
                bcf     PORTA, LED

cc0_1st_charge                                 ; CC0(du=+5.2v i=0.16mA:dt=7.15s) at same time
        CCpic(du=+2.7v i=0.16mA:dt=0.168sec)
                movlw   3                      ; CC0 charge time 7.15sec = 2.3sec *  3.1
                movwf   sleep_cnt
cc0_1st_charging                               ; WDT period=2.3s
                call    sleep_set
                sleep
                nop
                decfsz  sleep_cnt,f
                goto    cc0_1st_charging

                clrwdt                         ; clear WDT

                ; initialize program variables to zero
                movlw   0
                movwf   stat_group
                movwf   temperature
                movwf   synth_lock
                movwf   rssi
                movwf   reg_ctl
                movwf    tmr1_int_flg

                movlw   5                      ; initialize the pulse counter as 5 because we use
1kHz AM signal to test
                movlw   pulse_cnt              ; that 5ms burst is the button line to makes a nice
'beep'.

                clrwdt                         ; clear WDT


main_loop

enable_reg
                bsf        PORTC, REG_EN       ; Enable Voltage Regulator to drive RFIC
                clrwdt                         ; clear WDT

rf_transmit
                clrwdt                         ; clear WDT
                call    rf_tx                 ; send BEEP  cc0(dt=0.011s-0.013s i=21mA : du=-1.24v)
CCpic(dt=0.015s i=0.50mA : du=-0.75v)
                clrwdt                         ; clear WDT

disable_reg
                bcf             PORTC, REG_EN  ; Disable Voltage Regulator to stop discharge from
CC0

c_recharge; cc0(du=1.24v i=0.16mA : dt=1.7s) CCpic(du=0.75v i=0.16mA : dt=0.46s) WDT period=2.3s
                movlw   1                              ; Just only sleep 2.3s, if want more then
increase this value
                movwf   sleep_cnt
c_recharging
                call    sleep_set
                sleep
                nop
                decfsz  sleep_cnt, f
```

```
                goto    c_recharging
                clrwdt
                goto main_loop

; *************************
; initPIC
; Initialize I/O port direction, pullups, and other control registers
; *************************

initPIC

                bcf             STATUS, RP0     ; set to bank 0 (just in case)

                ; disable interrupts and disconnect  comparator
                bcf             INTCON, GIE     ; this is the default on PowerOnReset (POR), but
never hurts
                movlw   0x07                    ; disconnect comparator by setting CM2:CM0 high
                movwf   CMCON

                bsf     STATUS, RP0             ; set to bank 1 for the following high registers

                clrf    ANSEL                   ; set all pins (ports A and C) to be digital

                movlw   0x0F                    ; set port A pins to be inputs, except bit RA4, RA5,
                movwf   TRISA

                movlw   0xff                    ; enable all weak pullups (this is the default, so
not really needed)
                movwf   WPUA

                movlw    0x00                   ; disable PORTA pin change interrupt
                movwf   IOCA

                ; initialize port C
                movlw   0x00                    ; set all to outputs, except RC3 which is STATDAT
                movwf   TRISC

                ; set initial  values on ports
                bcf             STATUS, RP0     ; set back to bank 0

                ; set Port A and Port C
                ; Set Power Down RFIC, turn off LED, all set as 0 except RA3
                clrf    PORTA
                clrf    PORTC

                ; initialize Timer1 interrupt
                clrf    T1CON                   ; stop timer1, T1_OSC disabled, prescaler=1:1
                clrf    TMR1H
                clrf    TMR1L
                clrf    INTCON                  ; disable interrupts
                bsf         STATUS, RP0         ; set bank 1
                clrf    PIE1                    ; disable peripheral interrupts
                bcf             STATUS, RP0     ; set bank 0
                clrf    PIR1                    ; clear peripheral interrupts flag
                movlw    0x04                   ; internal clock, 1:1 prescaler, timer1 stop
                movwf    T1CON


                ; initialize interrupt control register
                movlw    b'11010000'            ; enable Gloable interrupts, enable Peripheral
interrupts, enable RA2 interrupts;
                movwf    INTCON

                ; set OPTION_REG at last step
                bsf         STATUS, RP0         ; set bank 1
                movlw   0x3f                    ; Interrupt on falling edge of RA2/INT pin;
Prescaler is assigned to the WDT; WDT Rate 2.3s=18ms*128
                movwf   OPTION_REG
                bcf             STATUS, RP0     ; set back to bank for normal use 0 !!!

                retlw   0
```

```
; *************************
; rf_tx
; Program RFIC to Transmit 5 burst
; *************************
rf_tx

rfic_pw_on
               bcf     PORTC, RFIC_PW_DN

pgm_standby
                       movlw B'00001000'      ; standby: pwr_sw_out_sw enable, PW_CTL_1 enable
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode

                       ; wait TCXO work stably for 3500us
                       movlw 0xF2             ; 3500us timing 0xf253 = 0xffff- 3500
                       movwf TMR1H

                       movlw 0x53
                       movwf TMR1L

                       call  tmr1_on          ; enable timer1

wait_tcxo
                       btfss tmr1_int_flg,0
                       goto  wait_tcxo
                       call  tmr1_off         ; disable timer1
                       bcf   tmr1_int_flg,0; clear tmr1_int_flg bit0

pgm_TX
                       ; 1st pulse of AM burst
                       movlw B'10001000'      ; enable LPA
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode
                       movlw B'00001000'      ; disable LPA
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode

                       ; 2nd pulse of AM burst
                       movlw B'10001000'      ; enable LPA
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode
                       movlw B'00001000'      ; disable LPA
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode

                       ; 3rd pulse of AM burst
                       movlw B'10001000'      ; enable LPA
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode
                       movlw B'00001000'      ; disable LPA
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode


                       ; 4th pulse of AM burst
                       movlw B'10001000'      ; enable LPA
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode
                       movlw B'00001000'      ; disable LPA
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode


                       ; 5th pulse of AM burst
                       movlw B'10001000'      ; enable LPA
                       movwf reg_ctl          ; initialize control byte
                       call  pgm_tx_mode
                       movlw B'00001000'      ; disable LPA
```

```
                    movwf reg_ctl         ; initialize control byte
                    call  pgm_tx_mode

rfic_pw_off
                    bsf   PORTC, RFIC_PW_DN ; Power off RFIC, disable PW_CTL_2

              return


; ************************
; outbits
; Shifts number of bits specified in 'numbits' from W into synthesizer.
; Data is output lsb first (right shifted)
; ************************

outbits       movwf   temp    ; save data passed in from W

outbits_loop  ; branch to set or clear bit depending on value of lsb
              btfsc   temp,0
              goto    outbits_set
              goto    outbits_clr

outbits_set   ; set or clear the data bit on the synth
              bsf             PORTC, PGMDAT
              goto    outbits_clk
outbits_clr   bcf     PORTC, PGMDAT

outbits_clk   ; toggle the clock line to input bit to synth
              bsf             PORTC, PGMCLK
              bcf             PORTC, PGMCLK

              ; process the next bit, or return if all done
              rrf             temp, F
              decfsz  numbits, F
              goto    outbits_loop
              retlw   0


; ************************
; latchbits
; latch the bits shifted in with outbits
; ************************

latchbits     ; Bring latch line high and then toggle the clock line
                    bsf             PORTC, LATCH
                    bsf             PORTC, PGMCLK
                    bcf             PORTC, PGMCLK
                    bcf             PORTC, LATCH
                    retlw   0


; ********************************
; pgm_tx_mode
; Programs synth, control, etc bits.
; See program header for explanation of bit pattern
; ********************************

pgm_tx_mode   ; Program 60 bits of control (see program header)

              ; 33 bits of VCO and synthesizer bits first...
              ; coarse tune
              movlw   4
              movwf   numbits
              movlw   B'1011' ; 420 to 455 MHz range
              call    outbits

              ; lower 8 bits of fractional count
              movlw   8
              movwf   numbits
              movlw   B'00000000'
              call    outbits
```

```
                ; upper 2 bits of fractional count
                movlw   2
                movwf   numbits
                movlw   B'00'
                call    outbits

                ; lower 8 bits of N count
                movlw   8
                movwf   numbits
                movlw   B'01011101'
                call    outbits

                ; upper 2 bits of N count
                movlw   2
                movwf   numbits
                movlw   B'00'
                call    outbits

                ; 7 bit ref divider
                movlw   7
                movwf   numbits
                movlw   B'0000100'
                call    outbits

                ; 2 bits of SDM control
                movlw   2
                movwf   numbits
                movlw   B'00'
                call    outbits

                ; now for the remaining 27 bits...
                ; all 14 attenuator bits = 0 (no attenuation)
                movlw   7
                movwf   numbits
                movlw   B'0000000'
                call    outbits
                movlw   7
                movwf   numbits
                movlw   B'0000000'
                call    outbits

                ; 8 bits of config control (LPAenabled)
                movlw   8
                movwf   numbits
                movf    reg_ctl,W               ; move reg_ctl content to W
                call    outbits

                ; 5 more bits (lna freq alignment=4, RSSIenable=0, and spare=0)
                movlw   5
                movwf   numbits
                movlw   B'01100'
                call    outbits

                ; latch the bits into the chip and return
                call    latchbits
                retlw   0

; ***********************************
; wait1ms
; Delays by about 1 milliseconds
; ***********************************
wait1ms movlw   1
                movwf   count           ; outer loop ms counter

wait1ms_outer   movlw   250
                movwf   temp            ; inner loop counter (250 times 4us = 1ms)

wait1ms_inner   decf    temp,F          ; dec inner counter -- 2us
                btfss   STATUS,Z        ; check zero flag  -- 2us
                goto    wait10ms_inner ; continue if not zero -- 4us
```

```
                decfsz  count, F       ; dec outer counter
                goto    wait10ms_outer ; continue if not zero

                retlw   0

; **********************************
; wait10ms
; Delays by about 10 milliseconds
; **********************************
wait10ms        movlw   10
                movwf   count          ; outer loop ms counter

wait10ms_outer movlw    250
                movwf   temp           ; inner loop counter (250 times 4us = 1ms)

wait10ms_inner decf     temp,F         ; dec inner counter -- 2us
                btfss   STATUS,Z       ; check zero flag  -- 2us
                goto    wait10ms_inner ; continue if not zero -- 4us

                decfsz  count, F       ; dec outer counter
                goto    wait10ms_outer ; continue if not zero

                retlw   0


; **********************************
; wait20ms
; Delays by about 100 milliseconds
; **********************************
wait20ms        movlw   20
                movwf   count          ; outer loop ms counter

wait20ms_outer movlw    250
                movwf   temp           ; inner loop counter (250 times 4us = 1ms)

wait20ms_inner decf     temp,F         ; dec inner counter -- 2us
                btfss   STATUS,Z       ; check zero flag  -- 2us
                goto    wait20ms_inner ; continue if not zero -- 4us

                decfsz  count, F       ; dec outer counter
                goto    wait20ms_outer ; continue if not zero

                retlw   0


; **********************************
; wait100ms
; Delays by about 100 milliseconds
; **********************************
wait100ms       movlw   100
                movwf   count          ; outer loop ms counter

wait100ms_outer         movlw   250
                movwf   temp           ; inner loop counter (250 times 4us = 1ms)

wait100ms_inner         decf    temp,F         ; dec inner counter -- 2us
                        btfss   STATUS,Z       ; check zero flag  -- 2us
                        goto    wait100ms_inner ; continue if not zero -- 4us

                        decfsz  count, F       ; dec outer counter
                        goto    wait100ms_outer        ; continue if not zero

                        retlw   0


; **********************************
; wait1sec
; Delays by about 1 second
; **********************************
```

```
wait1sec              call wait100ms
                      call wait100ms
                      call wait100ms
                      call wait100ms
                      call wait100ms
                      call wait100ms
                      call wait100ms
                      call wait100ms
                      call wait100ms
                      call wait100ms

                      retlw  0

; **********************************
; wait10sec
; Delays by about 10 second
; **********************************
wait10sec             call wait1sec
                      call wait1sec
                      call wait1sec
                      call wait1sec
                      call wait1sec
                      call wait1sec
                      call wait1sec
                      call wait1sec
                      call wait1sec
                      call wait1sec

                      retlw  0


; **********************************
; tmr1_on
; Enable timer1 with 100us waiting
;
; **********************************
tmr1_on
              ;movlw 0xFF                  ; 200us timing = 0xffff- 200= 0xff37
              ;movwf TMR1H

              ;movlw 0x37
              ;movwf TMR1L

              bsf      T1CON, TMR1ON       ; start timer1 on
              bsf      STATUS, RP0         ; set bank 1
              bsf      PIE1, TMR1IE        ; enable timer1
              bcf      STATUS, RP0         ; set bank 0

              return

; **********************************
; tmr1_off
; Disable timer1 with 100us waiting
;
; **********************************
tmr1_off

              bcf      T1CON, TMR1ON       ; stop timer1 on
              bsf      STATUS, RP0         ; set bank 1
              bcf      PIE1, TMR1IE        ; disable timer1
              bcf      STATUS, RP0         ; set bank 0

              return


; **********************************
; tmr1_timing
; Timer1 timing
; set timer to wait us = tmr1_int_cnt*0xffffus + (tmr1_H + tmrl_L)us
; **********************************
tmrl_timing
```

```
                        call   tmr1_on               ; enable timer1 as rollover running
wait_big_loop
                        btfss  tmr1_int_flg,0
                        goto   wait_big_loop

                        bcf    tmr1_int_flg,0        ; clear tmr1_int_flg bit0
                        decfsz tmr1_int_cnt,F        ; if 15 times over
                        goto   wait_big_loop
                        call   tmr1_off              ; disable timer1 to change counter

                        ; set sleep timer to wait the left time= tmrl_H + tmr1_L
                        movf   tmr1_H, W
                        movwf  TMR1H

                        movf   tmr1_L, W
                        movwf  TMR1L

                        call   tmr1_on               ; enable timer1

wait_small_loop
                        btfss  tmr1_int_flg,0
                        goto   wait_small_loop
                        call   tmr1_off              ; disable timer1

                        bcf    tmr1_int_flg,0        ; clear tmr1_int_flg bit0

                        return


; **********************************
; TMR1_INT
; ISR of timer1 overflow.
;
; **********************************
TMR1_INT
       bsf tmr1_int_flg,0    ; set tmr1_int_flg bit0 as 1
       bcf    PIR1, TMR1IF   ; clear the TIMER1 interrupt flag
       return                ; isr return

RA2_INT
       bcf INTCON, INTF      ; clear RA2 interrupt flag
       movlw 0x00
       movwf PCL


sleep_set
               movlw   0x00           ; DISABLE all weak pullups (this is the default, so not
really needed)
               movwf   WPUA

               movlw   0x07
               movwf   CMCON          ; Comparator OFF

               bcf     VRCON,   VREN ; CVref circuit power down
               bcf     ADCON0,  ADON  ; A/D Converter shut-off1

               return

;  that's all
       end
```

Appedix E - K-State EHR Demo Board Code for 400kHz CPU Clock

```
; Program peregine3# 433.92MHz TX.asm
;
; Test program for Peregine 1# board.
; By  Xiaohu Zhang, William B. Kuhn
; Created 08/23/07.
;
; This code is used to test the TX frequency. That means it will transmite
; always.
;
;
; TEST PLAN 2. The RFIC working depends on CC0, CCpic charge and discharge.
;*********************
; CC0   = 200uF
; CC1   = N/A
; CCpic = 10uF
;*********************
;Design:
;              Two charging capacitor are used here. One is CC0=220uF used to drive RFIC
;              working for 0.012s at least; another is CCpic=10uF used to drive Microchip
;              PIC16f676 are give power by 3.3v voltage regulator. So voltage regulator
;              are always working.
;              PIC16F676 operating specifications:
;                              Operating Frequency  :  400kH
;                              Oscillator Period    :  2.5uS
;                              Instruction Exe Time :  10uS = 4*2.5uS
;                              Operating Current    :  0.16mA
;                              Sleeping  Current    :
;                              TIMER1 Counte/time   :  10uS
;
; This program provides the 60 bits of programming needed to set
; the fab4 transceiver chip to transmitt mode with full gain and sleep mode
; enabled. To keep low power consuming, we use <1% duty cycle that means during
; 1 second, xx ms are occupied by RFIC to transmitt data and the remain time,
; xxx ms, RFIC will enter sleep mode to save power.
; The synthesizer is setup to transimitt a frequency close to CCSDS channel 1
; using integer-N mode.  The transmit freq is at 404.4 MHz.
;
; In the control field, LPAena is set to 1 to put the TR switch into transmit mode
; and Stby_bar is set to 0 to allow RFIC slepping.
;
; See other programs for details on the programming bits.


; Meta data for the compiler
       list    p = 16f676
       radix   dec
       include <p16f676.inc>
       errorlevel  -302             ; suppress arg range msg from list file

; Configure the chip for Code protect off, Data Code protect off, Brownout detect off, MCLR
disabled, WDT disabled, Powerup timer enabled, and internal oscillator (with no use of output pin)

    __CONFIG   _CP_OFF & _CPD_OFF & _BODEN_OFF & _MCLRE_OFF & _WDT_OFF & _PWRTE_ON &
_EXTRC_OSC_NOCLKOUT

; '__CONFIG' directive is used to embed configuration word within .asm file.
; The lables following the directive are located in the respective .inc file.
; See data sheet for additional information on configuration word settings.


; Symbolic constants

REG_EN                equ 0   ; RC0 (pin 10) ; Voltage regulator enable signal
```

```
STATCLK                 equ 1   ; RC1 (pin 9)  ; Serial programming input and status output
register control pins
LATCH                   equ 2   ; RC2 (pin 8)
;STATDAT                        equ 3   ; RC3 (pin 7)
STATDAT                 equ     3       ; RA3 (pin 4)  ; RA3 can only be used as INPUT
PGMCLK                  equ 4   ; RC4 (pin 6)
PGMDAT                  equ 5   ; RC5 (pin 5)

PW_CMP_H                equ 0    ; RA0 (pin 13)  ; Solar cell voltage compare
PW_CMP_L                equ 1    ; RA1 (pin 12)

PIC_RST                 equ 2    ; RA2 (pin 11)  ; PIC reset button
;RFIC_PW_DN             equ 3    ; RA3 (pin4)   ; RFIC Power Down pin
RFIC_PW_DN              equ 3    ; RC3 (pin 7)   ; RFIC Power Down pin
LED                     equ 4    ; RA4 (pin 3)   ; LED



; Register assignments

        cblock 0x20
w_temp                          ; temporary used in (unimplemented) ISR at 0x004
status_temp                     ; ditto
pclath_temp                     ; temporary used in ISR

numbits                         ; # of bits to write to rfic
temp                            ; temporary
count                           ; counter used in delay function

stat_group                      ; which status grouping to read / display
                                        ; 0 = temp, 1 = synthlock, 2 = rssi, 4 = spare

temperature                     ; variable to shift status bits into
synth_lock                      ; variable to shift status bits into
rssi                            ; variable to shift status bits into

pulse_cnt                       ; # of pulse send during wake up period, which is
                                        ; determined by the frequency of modulation signal

reg_ctl                         ; control subfield

tmr1_int_flg                    ; flag used to indicate whether timer interrupt occured
tmr1_int_cnt                    ; # of timer1 interrupts #*0xffff
tmr1_H                          ; the left timing number high byte
tmr1_L                          ; the left timing number low byte

sleep_cnt                       ; # of WDT sleep period depends on the WDT presca
        endc



; ******************  Start of program **************************

; Setup reset vector to skip over interrupt subroutines, if any
        org     0x00
                goto    start

; An interrupt service routine template.  (Not used in this program)
        org 0x004               ; interrupt vector location
                ; SAVE current status register
                movwf    w_temp           ; save off current W register contents
                movf    STATUS,w          ; move status register into W register
                movwf   status_temp       ; save off contents of STATUS register
                movf    PCLATH,w          ; move pclath register into w register
                movwf    pclath_temp      ; save off contents of PCLATH register


                ; isr code can go here or be located as a call subroutine elsewhere
                clrf    STATUS
                btfsc   PIR1, TMR1IF     ; timer1 overflow interrupt
```

```
                call    TMR1_INT         ; yes

                btfsc   INTCON, INTF     ; if RA2 pin interrupt occure
                call    RA2_INT          ; yes


                ; RESTORED status register
                movf    pclath_temp,w     ; retrieve copy of PCLATH register
                movwf   PCLATH            ; restore pre-isr PCLATH register contents
                movf    status_temp,w     ; retrieve copy of STATUS register
                movwf   STATUS           ; restore pre-isr STATUS register contents
                swapf   w_temp,f
                swapf   w_temp,w          ; restore pre-isr W register contents
                retfie                    ; return from interrupt


; ************************************
; The main routine
; ************************************

start
                ;  Calibration of internal oscillator (from 16f676 template code)
;               call    0x3FF             ; retrieve factory calibration value
                bsf     STATUS,RP0        ; set file register bank to 1
                call    0x3FF             ; retrieve factory calibration value
                movwf   OSCCAL            ; update register with factory cal value
                bcf     STATUS,RP0        ; set file register bank to 0

                ; initialize the PIC and its I/O ports
                call    initPIC

                bsf     PORTA, LED
                call    wait10ms
                bcf     PORTA, LED
                call    wait10ms
                bsf     PORTA, LED
                call    wait10ms
                bcf     PORTA, LED

                ; initialize program variables to zero
                movlw   0
                movwf   stat_group
                movwf   temperature
                movwf   synth_lock
                movwf   rssi
                movwf   reg_ctl
                movwf    tmr1_int_flg

                movlw   5; initialize the pulse counter as 5 because we use 1kHz AM signal to test
                movlw    pulse_cnt      ; that 5ms burst is the button line to makes a nice 'beep'.

                call    en_sw_software  ; enable software switch to enable vdd_vco

                call wait20ms

;pgm_TX

                movlw B'10001000'      ; enable LPA
                movwf reg_ctl          ; initialize control byte
                call  pgm_tx_mode
                bcf   PORTC, RFIC_PW_DN

main_loop

                bsf     PORTA, LED     ; Cannot be here because drop too much current
                call    wait100ms
                bcf     PORTA, LED
                call    wait100ms

                clrwdt                 ; clear WDT
```

```
                goto main_loop

; *************************
; initPIC
; Initialize I/O port direction, pullups, and other control registers
; *************************

initPIC

                bcf             STATUS, RP0     ; set to bank 0 (just in case)

                ; disable interrupts and disconnect  comparator
                bcf             INTCON, GIE     ; this is the default on PowerOnReset (POR), but
never hurts
                movlw   0x07                    ; disconnect comparator by setting CM2:CM0 high
                movwf   CMCON

                bsf             STATUS, RP0     ; set to bank 1 for the following high registers
                clrf    ANSEL                   ; set all pins (ports A and C) to be digital

                movlw   0x2F                    ; set port A pins to be inputs, except LED bit RA4,
                movwf   TRISA

                movlw   0xff                    ; enable all weak pullups (this is the default, so
not really needed)
                movwf   WPUA

                movlw    0x00                   ; disable PORTA pin change interrupt
                movwf   IOCA

                ; initialize port C
                movlw   0x00                    ; set all to outputs
                movwf   TRISC

                ; set initial  values on ports
                bcf             STATUS, RP0     ; set back to bank 0

                ; set Port A and Port C
                ;movlw   0x08                   ; Set Power Down RFIC, turn off LED, all set as 0
except RA3
                ;movlw   0x00
                ;movwf   PORTA
                clrf    PORTA
                movlw   0x08
                movwf   PORTC
                ;clrf   PORTC

                ; initialize Timer1 interrupt
                clrf    T1CON                   ; stop timer1, T1_OSC disabled, prescaler=1:1
                clrf    TMR1H
                clrf    TMR1L
                clrf    INTCON                  ; disable interrupts
                bsf     STATUS, RP0             ; set bank 1
                clrf    PIE1                    ; disable peripheral interrupts
                bcf     STATUS, RP0             ; set bank 0
                clrf    PIR1                    ; clear peripheral interrupts flag
                movlw   0x04                    ; internal clock, 1:1 prescaler, timer1 stop
                movwf   T1CON


                ; initialize interrupt control register
                movlw   b'11010000'             ; enable Gloable interrupts, enable Peripheral
interrupts, enable RA2 interrupts;
                movwf   INTCON

                ; set OPTION_REG at last step
                bsf         STATUS, RP0         ; set bank 1
                movlw   0x3E                    ; Interrupt on falling edge of RA2/INT pin;
Prescaler is assigned to the WDT; WDT Rate 1.15s=18ms*64
                movwf   OPTION_REG
                bcf     STATUS, RP0     ; set back to bank for normal use 0 !!!
```

```
              retlw   0

; *************************
; en_sw_software
; Enable software switch to enable vdd_vco
; *************************
en_sw_software
              movlw B'00001000'             ; enable standy_bar---enable sw_software
              movwf reg_ctl                 ; initialize control byte
              call  pgm_tx_mode
              return

; *************************
; dis_sw_software
; disable software switch to enable vdd_vco
; *************************
dis_sw_software
              movlw B'00000000'             ; enable standy_bar---enable sw_software
              movwf reg_ctl             ; initialize control byte
              call  pgm_tx_mode
              return


; *************************
; rf_tx
; Program RFIC to Transmit 5 burst
; *************************
rf_tx

pgm_TX
                      ; 1st pulse of AM burst
                      movlw B'10001000'      ; enable LPA

                      movwf reg_ctl          ; initialize control byte

                      call  pgm_tx_mode

                      ; wait TCXO work stably for 3500us=0x0DAC , due to Timer1 count cycle =
10uS
                      movlw 0xFE                    ; 3500us timing 0xfea1 = 0xffff- 350
                      movwf TMR1H
                      movlw 0xA1
                      movwf TMR1L

                      call  tmr1_on          ; enable timer1

wait_tcxo
                      btfss tmr1_int_flg,0
                      goto  wait_tcxo
                      call  tmr1_off         ; disable timer1
                      bcf   tmr1_int_flg,0; clear tmr1_int_flg bit0


TX_1st_beep
                      bcf   PORTC, RFIC_PW_DN
                      call  waitP5ms                ; wait 0.5mS for 1kHz AM beep
                      bsf   PORTC, RFIC_PW_DN ; Power off RFIC, disable PW_CTL_2
                      call  waitP5ms                ; wait 0.5mS for 1kHz AM beep


TX_2nd_beep
                      bcf   PORTC, RFIC_PW_DN
                      call  waitP5ms                ; wait 0.5mS for 1kHz AM beep
                      bsf   PORTC, RFIC_PW_DN ; Power off RFIC, disable PW_CTL_2
                      call  waitP5ms                ; wait 0.5mS for 1kHz AM beep


TX_3rd_beep
                      bcf   PORTC, RFIC_PW_DN
                      call  waitP5ms                ; wait 0.5mS for 1kHz AM beep
```

```
                        bsf   PORTC, RFIC_PW_DN       ; Power off RFIC, disable PW_CTL_2
                        call  waitP5ms                ; wait 0.5mS for 1kHz AM beep

TX_4th_beep
                        bcf   PORTC, RFIC_PW_DN
                        call  waitP5ms                ; wait 0.5mS for 1kHz AM beep
                        bsf   PORTC, RFIC_PW_DN ; Power off RFIC, disable PW_CTL_2
                        call  waitP5ms                ; wait 0.5mS for 1kHz AM beep


TX_5th_beep
                        bcf   PORTC, RFIC_PW_DN
                        call  waitP5ms                ; wait 0.5mS for 1kHz AM beep
                        bsf   PORTC, RFIC_PW_DN       ; Power off RFIC, disable PW_CTL_2
                        ;call  waitP5ms               ; wait 0.5mS for 1kHz AM beep

                return


; ************************
; outbits
; Shifts number of bits specified in 'numbits' from W into synthesizer.
; Data is output lsb first (right shifted)
; ************************

outbits         movwf   temp    ; save data passed in from W

outbits_loop    ; branch to set or clear bit depending on value of lsb
                btfsc   temp,0
                goto    outbits_set
                goto    outbits_clr

outbits_set     ; set or clear the data bit on the synth
                bsf             PORTC, PGMDAT
                goto    outbits_clk
outbits_clr     bcf     PORTC, PGMDAT

outbits_clk     ; toggle the clock line to input bit to synth
                bsf             PORTC, PGMCLK
                bcf             PORTC, PGMCLK

                ; process the next bit, or return if all done
                rrf             temp, F
                decfsz  numbits, F
                goto    outbits_loop
                retlw   0


; ************************
; latchbits
; latch the bits shifted in with outbits
; ************************

latchbits       ; Bring latch line high and then toggle the clock line
                        bsf             PORTC, LATCH
                        bsf             PORTC, PGMCLK
                        bcf             PORTC, PGMCLK
                        bcf             PORTC, LATCH
                        retlw   0


; ********************************
; pgm_tx_mode
; Programs synth, control, etc bits.
; See program header for explanation of bit pattern
; ********************************

pgm_tx_mode     ; Program 60 bits of control (see program header)

                ; 33 bits of VCO and synthesizer bits first...
                ; coarse tune
```

```
                movlw   4
                movwf   numbits
                ;movlw  B'1011' ; 420 to 455 MHz range
                movlw   B'1011' ; 400 to 445 MHz range
                call    outbits

                ; lower 8 bits of fractional count
                movlw   8
                movwf   numbits
                movlw   B'10011010'  ; 433.92MHz  410  -->432.018
                call    outbits

                ; upper 2 bits of fractional count
                movlw   2
                movwf   numbits
                movlw   B'01'; 433.92MHz
                call    outbits

                ; lower 8 bits of N count
                movlw   8
                movwf   numbits
                movlw   B'01011010' ; 433.92MHz  90
                call    outbits

                ; upper 2 bits of N count
                movlw   2
                movwf   numbits
                movlw   B'00'
                call    outbits

                ; 7 bit ref divider
                movlw   7
                movwf   numbits
                movlw   B'0000100'
                call    outbits

                ; 2 bits of SDM control
                movlw   2
                movwf   numbits
                movlw   B'11'
                call    outbits

                ; now for the remaining 27 bits...
                ; all 14 attenuator bits = 0 (no attenuation)
                movlw   7
                movwf   numbits
                movlw   B'0000000'
                call    outbits
                movlw   7
                movwf   numbits
                movlw   B'0000000'
                call    outbits

                ; 8 bits of config control (LPAenabled)
                movlw   8
                movwf   numbits
                movf    reg_ctl,W             ; move reg_ctl content to W
                call    outbits

                ; 5 more bits (lna freq alignment=4, RSSIenable=0, and spare=0)
                movlw   5
                movwf   numbits
                movlw   B'01100'
                call    outbits


                ; latch the bits into the chip and return
                call    latchbits
                retlw   0

; ***********************************
```

```
; wait0.5ms
; Delays by about 0.5 milliseconds
; **********************************
waitP5ms              movlw   12
                      movwf   temp           ; inner loop counter (12.5 * 40us = 0.5ms)

waitP5ms_inner decf   temp,F         ; dec inner counter -- 20us
                      btfss   STATUS,Z       ; check zero flag  -- 20us
                      goto    waitP5ms_inner ; continue if not zero -- 40us

                      retlw   0

; **********************************
; wait1ms
; Delays by about 1 milliseconds
; **********************************
wait1ms               movlw   25
                      movwf   temp           ; inner loop counter (25 times 40us = 1ms)

wait1ms_inner  decf   temp,F         ; dec inner counter -- 20us
                      btfss   STATUS,Z       ; check zero flag  -- 20us
                      goto    wait1ms_inner ; continue if not zero -- 40us

                      retlw   0

; **********************************
; wait10ms
; Delays by about 10 milliseconds
; **********************************
wait10ms              movlw   250
                      movwf   temp           ; inner loop counter (250 times 40us = 10ms)

wait10ms_inner decf   temp,F         ; dec inner counter -- 20us
                      btfss   STATUS,Z       ; check zero flag  -- 20us
                      goto    wait10ms_inner ; continue if not zero -- 40us

                      retlw   0


; **********************************
; wait20ms
; Delays by about 20 milliseconds
; **********************************
wait20ms       movlw   2
                      movwf   count          ; outer loop ms counter

wait20ms_outer movlw   250
                      movwf   temp           ; inner loop counter (250 times 40us = 10ms)

wait20ms_inner decf   temp,F         ; dec inner counter -- 20us
                      btfss   STATUS,Z       ; check zero flag  -- 20us
                      goto    wait20ms_inner ; continue if not zero -- 40us

                      decfsz  count, F       ; dec outer counter
                      goto    wait20ms_outer ; continue if not zero

                      retlw   0


; **********************************
; wait100ms
; Delays by about 100 milliseconds
; **********************************
wait100ms      movlw   10
                      movwf   count          ; outer loop ms counter

wait100ms_outer       movlw   250
                      movwf   temp           ; inner loop counter (250 times 40us = 10ms)

wait100ms_inner       decf    temp,F         ; dec inner counter -- 20us
                      btfss   STATUS,Z       ; check zero flag  -- 20us
```

```
                        goto    wait100ms_inner ; continue if not zero -- 40us

                        decfsz  count, F        ; dec outer counter
                        goto    wait100ms_outer         ; continue if not zero

                        retlw   0


; **********************************
; wait1sec
; Delays by about 1 second
; **********************************
wait1sec                call wait100ms
                        call wait100ms
                        call wait100ms
                        call wait100ms
                        call wait100ms
                        call wait100ms
                        call wait100ms
                        call wait100ms
                        call wait100ms
                        call wait100ms

                        retlw   0

; **********************************
; wait10sec
; Delays by about 10 second
; **********************************
wait10sec               call wait1sec
                        call wait1sec
                        call wait1sec
                        call wait1sec
                        call wait1sec
                        call wait1sec
                        call wait1sec
                        call wait1sec
                        call wait1sec
                        call wait1sec

                        retlw   0


; **********************************
; tmr1_on
; Enable timer1 with 100us waiting
;
; **********************************
tmr1_on
                ;movlw 0xFF                             ; 200us timing = 0xffff- 200= 0xff37
                ;movwf TMR1H

                ;movlw 0x37
                ;movwf TMR1L

                bsf       T1CON, TMR1ON                 ; start timer1 on
                bsf       STATUS, RP0           ; set bank 1
                bsf       PIE1, TMR1IE          ; enable timer1
                bcf       STATUS, RP0           ; set bank 0

                return

; **********************************
; tmr1_off
; Disable timer1 with 100us waiting
;
; **********************************
tmr1_off

                bcf       T1CON, TMR1ON                 ; stop timer1 on
```

```
                bsf     STATUS, RP0         ; set bank 1
                bcf     PIE1, TMR1IE        ; disable timer1
                bcf     STATUS, RP0         ; set bank 0

                return


; **********************************
; tmr1_timing
; Timer1 timing
; set timer to wait us = tmr1_int_cnt*0xffffus + (tmr1_H + tmrl_L)us
; **********************************
tmrl_timing
                        call   tmr1_on              ; enable timer1 as rollover running
wait_big_loop
                        btfss  tmr1_int_flg,0
                        goto   wait_big_loop

                        bcf    tmr1_int_flg,0       ; clear tmr1_int_flg bit0
                        decfsz tmr1_int_cnt,F       ; if 15 times over
                        goto   wait_big_loop
                        call   tmr1_off             ; disable timer1 to change counter

                        ; set sleep timer to wait the left time= tmrl_H + tmr1_L
                        movf   tmr1_H, W
                        movwf  TMR1H

                        movf   tmr1_L, W
                        movwf  TMR1L

                        call   tmr1_on              ; enable timer1

wait_small_loop
                        btfss  tmr1_int_flg,0
                        goto   wait_small_loop
                        call   tmr1_off             ; disable timer1
                        bcf    tmr1_int_flg,0       ; clear tmr1_int_flg bit0

                        return


; **********************************
; TMR1_INT
; ISR of timer1 overflow.
;
; **********************************
TMR1_INT

       bsf tmr1_int_flg,0    ; set tmr1_int_flg bit0 as 1
       bcf     PIR1, TMR1IF  ; clear the TIMER1 interrupt flag
       return                ; isr return

RA2_INT
       bcf INTCON, INTF      ; clear RA2 interrupt flag
       movlw 0x00
       movwf PCL

sleep_set
                movlw   0x00; DISABLE all weak pullups (this is the default, so not really needed)
                movwf   WPUA
                movlw    0x07
                movwf    CMCON          ;Comparator OFF
                bcf      VRCON,   VREN   ; CVref circuit power down
                bcf      ADCON0,  ADON    ; A/D Converter shut-off1
                return

;  that's all

       End
```

Appedix F - Frequency Synthesizer PIC12F509 Code

F.1 151MHz Board

```
; Frequency synthesizer controller for EECE662/664 GOES Satellite Downconverter
; By           William B. Kuhn
; Created      11/25/00
; Revised      12/01/06 for use with 12F509
; Revised   8/17/07 for downconversion of LRIT signal to 138.000 MHz

; Modified  Xiaohu Zhang  5/22/2008
; Modified to be used for 151.94MHz transmiter
;      f_vco = 151.94e6 = [32*B + A] * (19.2e6/R)  --> 151.94e6 = (N*19.2e6)/R
;      when N = 7597 then R = 960
;          thus B = 237, A = 13 and R = 960
;
;      N-reg :
;     LSB   |--- A ---|---------------  B ---------------| MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       1  0 1 0 1 1 0 1 0 1 1  0  1  1  1  0  0  0  0  0  0
;
;      R-reg
;     LSB   |----------------  R   ----------|-- TEST ---|MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       0  0 0 0 0 0 0 0 1 1 1  1  0  0  0  0  0  0  0  0  1


; This program sets up the National Semiconductor LMX2326 synthesizer chip
; and increments through channels when a switch is pressed.
; The synthesizer is programmed for a reference frequency of 500 kHz (1/8 of 4 MHz
; clock), and programs one of 10 channels from 138 MHz to 147 MHz.
; On powerup, the channel is set to 138 MHz.  Each press of the channel up
; switch will advance to the next channel, 1 MHz higher.  After channel 10
; is reached (147 MHz), the synth is programmed to an invalid channel to force
; the lock light to go out (useful for testing and also for knowing where one
; is in the sequence).  At the next press, channel 1 is again programmed
; and the sequence repeats.



; Meta data for the compiler
; This is for a 12f509 with 1024 words of program and 41 file registers
; However, for ease of use (no bank switching), we will limit code to 512 words and 25 file regs
       list    p = 12f509
       radix   dec
       include <p12f509.inc>

; Configure the chip for MCLR disabled, Code protect off,
; WDT disabled, and crystal oscillator
       __config       H'0a'

; Symbolic constants

NUMCHAN        equ     20      ; number of channels we can tune to
CLKBIT         equ     0       ; GPIO bit 0 is clock bit on synth
DATABIT        equ     1       ; GPIO bit 1 is data bit on synth
LATCHBIT       equ     2       ; GPIO bit 2 is LE (latch) bit on synth
SWITCHBIT      equ     3       ; GPIO bit 3 is channel up switch

LED1           equ 5   ; GPIO bit 5 is LED1

; Register assignments

channel        equ     8       ; current channel
numbits        equ     9       ; # of bits to write to synth
Ndiv           equ     10      ; low order byte of N reg
```

```
temp            equ     11      ; temporary
count           equ     12      ; counter for delay loop


; ******************  Start of program *************************

; Setup reset vector to skip over subroutines, which must reside
; in first 256 addresses
        org     0
        goto    start

; *************************
; initGPIO
; Initialize I/O port direction and set bits to zero
; *************************

initGPIO        ; first set the option register bits to allow GP2 to be an output
                movlw   B'10001111'
                option

                ; next, set the tristate register to enable GP0 -> GP2 and GP5 as outputs
                movlw   B'11011000'
                tris    6

                ; clear the outputs (probably redundant with reset state, but hey...)
                bcf     GPIO, DATABIT
                bcf     GPIO, CLKBIT
                bcf     GPIO, LATCHBIT
                ;bcf GPIO, SWITCHBIT

                retlw   0

; ************************
; outbits
; Shifts number of bits specified in 'numbits' from W into synthesizer.
; Data is output msb first (left shifted)
; ************************

outbits         movwf   temp    ; save data passed in from W

outbits_loop    ; branch to set or clear bit depending on value of msb
                btfsc   temp,7
                goto    outbits_set
                goto    outbits_clr

outbits_set     ; set or clear the data bit on the synth
                bsf     GPIO, DATABIT
                goto    outbits_clk
outbits_clr     bcf     GPIO, DATABIT

outbits_clk     ; toggle the clock line to input bit to synth
                bsf     GPIO, CLKBIT
                bcf     GPIO, CLKBIT

                ; process the next bit, or return if all done
                rlf     temp, F
                decfsz  numbits, F
                goto    outbits_loop
                retlw   0

; ************************
; latchbits
; toggles the LE line on the synth to latch the bits shifted in with outbits
; ************************

latchbits       bsf     GPIO, LATCHBIT
                bcf     GPIO, LATCHBIT
                retlw   0


; *******************************
```

```
; initF
; Initialize the F register (programming reg) on the synth.
; VCO slope as + (F6=1), digital lock detect is selected (F5:F3=001),
; Powerdown/reset is disabled (F2:F1 = 00), and the initialization mode is
; selected (C2:C1 = 11).
; *********************************

initF
               ; F19
               movlw  1
               movwf  numbits
               clrw
               call   outbits

               ; F18:F11 = 0
               movlw   8
               movwf   numbits
               clrw
               call    outbits

               ; F10:F7 = 0, F6:F3 set as explained above
               movlw   8
               movwf   numbits
               ;movlw  B'00001001'
               movlw   B'00001000'    ;f3--f5--110 active high
               call    outbits

               ; F2:F1 and C2:C1 set as explained above
               movlw   4
               movwf   numbits
               movlw   B'00110000'
               call    outbits

               ; latch the bits into the synthesizer and return
               call    latchbits
               retlw   0


; ******************************
; initR
; R = 96 = 0110 0000
; Initialize the R register in the synth to divide by 960 (19.2 MHz ->  80kHz)
;
;      R-reg
;     LSB   |----------------  R   ----------|-- TEST ---|MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       0  0 0 0 0 0 0 1 1 0 0  0  0  0  0  0  0  0  0  0  1

initR          ; begin by writing top 5 bits to register
               ; (sets lock-detect precision to one and zeros test mode bits)
               movlw   5
               movwf   numbits
               movlw   H'80'
               call    outbits

               ; next, write R14:R7
               movlw   8
               movwf   numbits
               movlw    b'00000001'
               call    outbits

               ; write R6:R1
               movlw   6
               movwf   numbits
               movlw    b'10000000'
               call    outbits

               ; finally, write two lsb's as zero to designate R register
               movlw   2
               movwf   numbits
               clrw
```

```
                call    outbits

                ; latch the bits into the synth and return
                call    latchbits
                retlw   0

; ************************
; inc_chan
; Increments the current channel, wrapping to zero if max channel reached
; ************************

inc_chan        incf    channel, F
                movlw   NUMCHAN + 1
                subwf   channel, W
                btfsc   STATUS, Z

                clrf    channel
                retlw   0

; *********************************
; writeN
; N = 759.5 --> B=23  A=22
; 151.94e9 = [32*B + A] * (19.2e6/96)
;
;      N-reg :
;     LSB   |--- A ---|---------------  B ---------------| MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       1  0 0 1 1 0 1 1 1 1 0  1  0  0  0  0  0  0  0  0  0
writeN
                ;msb
                movlw  1
                movwf  numbits
                clrw
                call   outbits

                ; write first 8 bits of B N18:N11
                movlw  8
                movwf  numbits
                movlw   b'00000000'
                call    outbits

                ; write last 5 bits of B  N10:N6
                movlw  5
                movwf  numbits
                movlw   b'10111000'
                call    outbits

                ; write last 5 bits of A  N5:N1
                movlw  5
                movwf  numbits
                movlw   b'10110000'
                call    outbits

                ; write two control bits designating this as N register
                movlw  2
                movwf  numbits
                movlw  H'40'
                call    outbits

                ; latch into synth and return
                call    latchbits
                retlw   0

; *********************************
; wait250ms
; Delays by about 250 milliseconds
; *********************************

wait250ms       movlw   250
                movwf   count          ; outer loop ms counter
```

```
wait250ms_outer       movlw  250
              movwf   temp           ; inner loop counter (250 times 4us = 1ms)

wait250ms_inner       decf   temp,F         ; dec inner counter -- 1us
              btfss   STATUS,Z       ; check zero flag  -- 1us
              goto    wait250ms_inner ; continue if not zero -- 2us

              decfsz  count, F       ; dec outer counter
              goto    wait250ms_outer        ; continue if not zero

              retlw   0


; ************************************
; The main routine
; ************************************

start         ; initialize the PIC and its I/O ports
              call    initGPIO

              ; initialize the synthesizer and channel number
              call    initF          ; initialize the programming setup regs
              call    initR          ; initialize the ref freq divisor
              call    writeN         ; write the N divisor for this channel

              ; clear output (especially for when the last bit is '1')
              bcf     GPIO, DATABIT
              bcf     GPIO, CLKBIT
              bcf     GPIO, LATCHBIT

              bsf            GPIO,LED1
              call    wait250ms
              call    wait250ms
              bcf     GPIO,LED1

main_loop
; loop, checking channel switch for press (SWITCHBIT = 0)

              btfsc   GPIO, SWITCHBIT
              goto    main_loop

              call    writeN         ; write the N divisor for this channel

              bcf     GPIO, DATABIT
              bcf     GPIO, CLKBIT
              bcf     GPIO, LATCHBIT

              ; clear output (especially for when the last bit is '1')
              bsf            GPIO,LED1
              call    wait250ms
              bcf     GPIO,LED1

              goto main_loop

       end
```

F.2 433MHz Board

```
; Frequency synthesizer controller for EECE662/664 GOES Satellite Downconverter
; By           William B. Kuhn
; Created      11/25/00
; Revised      12/01/06 for use with 12F509
; Revised   8/17/07 for downconversion of LRIT signal to 138.000 MHz

; Modified  Xiaohu Zhang  5/22/2008
; Modified to be used for 433.92MHz transmiter
;      f_vco = 433.92e6 = [32*B + A] * (19.2e6/R)  --> 433.92e6 = (N*19.2e6)/R
;      when N = 1130 then R = 50
;          thus B = 35, A = 10 and R = 50
;
;      N-reg :
;     LSB   |--- A ---|---------------  B ---------------| MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       1  0 0 1 0 1 0 1 1 0 0  0  1  0  0  0  0  0  0  0
;
;      R-reg
;     LSB   |----------------  R   ----------|-- TEST ---|MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       0  0 0 1 0 0 1 1 0 0 0  0  0  0  0  0  0  0  0  1


; This program sets up the National Semiconductor LMX2326 synthesizer chip
; and increments through channels when a switch is pressed.
; The synthesizer is programmed for a reference frequency of 500 kHz (1/8 of 4 MHz
; clock), and programs one of 10 channels from 138 MHz to 147 MHz.
; On powerup, the channel is set to 138 MHz.  Each press of the channel up
; switch will advance to the next channel, 1 MHz higher.  After channel 10
; is reached (147 MHz), the synth is programmed to an invalid channel to force
; the lock light to go out (useful for testing and also for knowing where one
; is in the sequence).  At the next press, channel 1 is again programmed
; and the sequence repeats.


; Meta data for the compiler
; This is for a 12f509 with 1024 words of program and 41 file registers
; However, for ease of use (no bank switching), we will limit code to 512 words and 25 file regs
       list    p = 12f509
       radix   dec
       include <p12f509.inc>

; Configure the chip for MCLR disabled, Code protect off,
; WDT disabled, and crystal oscillator
       __config       H'0a'

; Symbolic constants

NUMCHAN        equ     20      ; number of channels we can tune to
CLKBIT         equ     0       ; GPIO bit 0 is clock bit on synth
DATABIT        equ     1       ; GPIO bit 1 is data bit on synth
LATCHBIT       equ     2       ; GPIO bit 2 is LE (latch) bit on synth
SWITCHBIT      equ     3       ; GPIO bit 3 is channel up switch

LED1           equ 5   ; GPIO bit 5 is LED1

; Register assignments

channel        equ     8       ; current channel
numbits        equ     9       ; # of bits to write to synth
Ndiv           equ     10      ; low order byte of N reg
temp           equ     11      ; temporary
count          equ     12      ; counter for delay loop



; *****************  Start of program *************************
```

```
; Setup reset vector to skip over subroutines, which must reside
; in first 256 addresses
       org     0
       goto    start

; *************************
; initGPIO
; Initialize I/O port direction and set bits to zero
; *************************

initGPIO       ; first set the option register bits to allow GP2 to be an output
               movlw   B'10001111'
               option

               ; next, set the tristate register to enable GP0 -> GP2 and GP5 as outputs
               movlw   B'11011000'
               tris    6

               ; clear the outputs (probably redundant with reset state, but hey...)
               bcf     GPIO, DATABIT
               bcf     GPIO, CLKBIT
               bcf     GPIO, LATCHBIT
               ;bcf GPIO, SWITCHBIT

               retlw   0

; ************************
; outbits
; Shifts number of bits specified in 'numbits' from W into synthesizer.
; Data is output msb first (left shifted)
; ************************

outbits        movwf   temp    ; save data passed in from W

outbits_loop   ; branch to set or clear bit depending on value of msb
               btfsc   temp,7
               goto    outbits_set
               goto    outbits_clr

outbits_set    ; set or clear the data bit on the synth
               bsf     GPIO, DATABIT
               goto    outbits_clk
outbits_clr    bcf     GPIO, DATABIT

outbits_clk    ; toggle the clock line to input bit to synth
               bsf     GPIO, CLKBIT
               bcf     GPIO, CLKBIT

               ; process the next bit, or return if all done
               rlf     temp, F
               decfsz  numbits, F
               goto    outbits_loop
               retlw   0

; ************************
; latchbits
; toggles the LE line on the synth to latch the bits shifted in with outbits
; ************************

latchbits      bsf     GPIO, LATCHBIT
               bcf     GPIO, LATCHBIT
               retlw   0


; *******************************
; initF
; Initialize the F register (programming reg) on the synth.
; VCO slope as + (F6=1), digital lock detect is selected (F5:F3=001),
; Powerdown/reset is disabled (F2:F1 = 00), and the initialization mode is
; selected (C2:C1 = 11).
```

```
; *********************************

initF
               ; F19
               movlw  1
               movwf  numbits
               clrw
               call   outbits

               ; F18:F11 = 0
               movlw  8
               movwf  numbits
               clrw
               call   outbits

               ; F10:F7 = 0, F6:F3 set as explained above
               movlw  8
               movwf  numbits
               ;movlw B'00001001'
               movlw  B'00001000'    ;f3--f5--110 active high
               call   outbits

               ; F2:F1 and C2:C1 set as explained above
               movlw  4
               movwf  numbits
               movlw  B'00110000'
               call   outbits

               ; latch the bits into the synthesizer and return
               call   latchbits
               retlw  0


; ******************************
; initR
; Initialize the R register in the synth to divide by 50 (19.2 MHz ->  384kHz)
;
;      R-reg
;     LSB   |----------------  R   ----------|-- TEST ---|MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       0  0 0 1 0 0 1 1 0 0 0  0  0  0  0  0  0  0  0  0  1

initR          ; begin by writing top 5 bits to register
               ; (sets lock-detect precision to one and zeros test mode bits)
               movlw  5
               movwf  numbits
               movlw  H'80'
               call   outbits

               ; next, write R14:R7 divisor bits as all zeros
               movlw  8
               movwf  numbits
               movlw   b'00000000'
               call   outbits

               ; write R6:R1 = 08h to give divide by 8
               movlw  6
               movwf  numbits
               movlw   b'11001000'
               call   outbits

               ; finally, write two lsb's as zero to designate R register
               movlw  2
               movwf  numbits
               clrw
               call   outbits

               ; latch the bits into the synth and return
               call   latchbits
               retlw  0
```

```
; ***********************
; inc_chan
; Increments the current channel, wrapping to zero if max channel reached
; ***********************

inc_chan       incf    channel, F
               movlw   NUMCHAN + 1
               subwf   channel, W
               btfsc   STATUS, Z

               clrf    channel
               retlw   0


; *********************************
; writeN
; N = 1130 --> B=35  A=10
; 433.92e9 = [32*B + A] * (19.2e6/50)
;
;      N-reg :
;     LSB   |--- A ---|---------------  B ---------------| MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       1  0 0 1 0 1 0 1 1 0 0  0  1  0  0  0  0  0  0  0  0
writeN
               ;msb
               movlw  1
               movwf  numbits
               clrw
               call   outbits

               ; write first 8 bits of B N18:N11
               movlw   8
               movwf   numbits
               movlw    b'00000001'
               call    outbits

               ; write last 5 bits of B  N10:N6
               movlw   5
               movwf   numbits
               movlw    b'00011000'
               call    outbits

               ; write last 5 bits of A  N5:N1
               movlw   5
               movwf   numbits
               movlw    b'01010000'
               call    outbits

               ; write two control bits designating this as N register
               movlw   2
               movwf   numbits
               movlw   H'40'
               call    outbits

               ; latch into synth and return
               call    latchbits
               retlw   0

; **********************************
; wait250ms
; Delays by about 250 milliseconds
; **********************************

wait250ms      movlw   250
               movwf   count          ; outer loop ms counter

wait250ms_outer        movlw   250
               movwf   temp           ; inner loop counter (250 times 4us = 1ms)

wait250ms_inner        decf    temp,F         ; dec inner counter -- 1us
               btfss   STATUS,Z       ; check zero flag  -- 1us
```

```
                goto    wait250ms_inner ; continue if not zero -- 2us

                decfsz  count, F        ; dec outer counter
                goto    wait250ms_outer         ; continue if not zero

                retlw   0


; ************************************
; The main routine
; ************************************

start           ; initialize the PIC and its I/O ports
                call    initGPIO

                ; initialize the synthesizer and channel number
                call    initF           ; initialize the programming setup regs
                call    initR           ; initialize the ref freq divisor
                call    writeN          ; write the N divisor for this channel

                ; clear output (especially for when the last bit is '1')
                bcf     GPIO, DATABIT
                bcf     GPIO, CLKBIT
                bcf     GPIO, LATCHBIT

                bsf             GPIO,LED1
                call    wait250ms
                call    wait250ms
                bcf      GPIO,LED1

main_loop
; loop, checking channel switch for press (SWITCHBIT = 0)


                btfsc   GPIO, SWITCHBIT
                goto     main_loop

                call    writeN          ; write the N divisor for this channel

                bcf     GPIO, DATABIT
                bcf     GPIO, CLKBIT
                bcf     GPIO, LATCHBIT

                ; clear output (especially for when the last bit is '1')
                bsf     GPIO,LED1
                call    wait250ms
                bcf      GPIO,LED1

                goto main_loop

        end
```

F.3 902MHz Board

```
; Frequency synthesizer controller for EECE662/664 GOES Satellite Downconverter
; By           William B. Kuhn
; Created      11/25/00
; Revised      12/01/06 for use with 12F509
; Revised   8/17/07 for downconversion of LRIT signal to 138.000 MHz

; Modified  Xiaohu Zhang
; Modified to be used for 902MHz transmiter
;      f_vco = 902e6 = [32*B + A] * (19.2e6/R)  --> 902e6 = (N*19.2e6)/R
;      when N = 2255 then R = 48
;          thus B = 70, A = 15 and R = 48
;
;      N-reg :
;     LSB   |--- A ---|---------------  B ---------------| MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       1  0 1 1 1 0 0 0 1 1 0  0  0  1  0  0  0  0  0  0  0
;
;      R-reg
;     LSB   |----------------  R   ----------|-- TEST ---|MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       0  0 0 0 0 0 1 1 0 0 0  0  0  0  0  0  0  0  0  0  1


; This program sets up the National Semiconductor LMX2326 synthesizer chip
; and increments through channels when a switch is pressed.
; The synthesizer is programmed for a reference frequency of 500 kHz (1/8 of 4 MHz
; clock), and programs one of 10 channels from 138 MHz to 147 MHz.
; On powerup, the channel is set to 138 MHz.  Each press of the channel up
; switch will advance to the next channel, 1 MHz higher.  After channel 10
; is reached (147 MHz), the synth is programmed to an invalid channel to force
; the lock light to go out (useful for testing and also for knowing where one
; is in the sequence).  At the next press, channel 1 is again programmed
; and the sequence repeats.


; Meta data for the compiler
; This is for a 12f509 with 1024 words of program and 41 file registers
; However, for ease of use (no bank switching), we will limit code to 512 words and 25 file regs
       list    p = 12f509
       radix   dec
       include <p12f509.inc>

; Configure the chip for MCLR disabled, Code protect off,
; WDT disabled, and crystal oscillator
       __config       H'0a'


; Symbolic constants

NUMCHAN        equ     20      ; number of channels we can tune to
CLKBIT         equ     0       ; GPIO bit 0 is clock bit on synth
DATABIT        equ     1       ; GPIO bit 1 is data bit on synth
LATCHBIT       equ     2       ; GPIO bit 2 is LE (latch) bit on synth
SWITCHBIT      equ     3       ; GPIO bit 3 is channel up switch

LED1           equ 5   ; GPIO bit 5 is LED1

; Register assignments

channel        equ     8       ; current channel
numbits        equ     9       ; # of bits to write to synth
Ndiv           equ     10      ; low order byte of N reg
temp           equ     11      ; temporary
count          equ     12      ; counter for delay loop


; *****************  Start of program *************************
```

```
; Setup reset vector to skip over subroutines, which must reside
; in first 256 addresses
       org    0
       goto   start

; *************************
; initGPIO
; Initialize I/O port direction and set bits to zero
; *************************

initGPIO       ; first set the option register bits to allow GP2 to be an output
               movlw  B'10001111'
               option

               ; next, set the tristate register to enable GP0 -> GP2 and GP5 as outputs
               movlw  B'11011000'
               tris   6

               ; clear the outputs (probably redundant with reset state, but hey...)
               bcf    GPIO, DATABIT
               bcf    GPIO, CLKBIT
               bcf    GPIO, LATCHBIT
               ;bcf GPIO, SWITCHBIT

               retlw  0

; ************************
; outbits
; Shifts number of bits specified in 'numbits' from W into synthesizer.
; Data is output msb first (left shifted)
; ************************

outbits        movwf  temp   ; save data passed in from W

outbits_loop   ; branch to set or clear bit depending on value of msb
               btfsc  temp,7
               goto   outbits_set
               goto   outbits_clr

outbits_set    ; set or clear the data bit on the synth
               bsf    GPIO, DATABIT
               goto   outbits_clk
outbits_clr    bcf    GPIO, DATABIT

outbits_clk    ; toggle the clock line to input bit to synth
               bsf    GPIO, CLKBIT
               bcf    GPIO, CLKBIT

               ; process the next bit, or return if all done
               rlf    temp, F
               decfsz numbits, F
               goto   outbits_loop
               retlw  0

; ************************
; latchbits
; toggles the LE line on the synth to latch the bits shifted in with outbits
; ************************

latchbits      bsf    GPIO, LATCHBIT
               bcf    GPIO, LATCHBIT
               retlw  0


; *********************************
; initF
; Initialize the F register (programming reg) on the synth.
; VCO slope as + (F6=1), digital lock detect is selected (F5:F3=001),
; Powerdown/reset is disabled (F2:F1 = 00), and the initialization mode is
; selected (C2:C1 = 11).
; *********************************
```

```
initF
               ; F19
               movlw  1
               movwf  numbits
               clrw
               call   outbits

               ; F18:F11 = 0
               movlw   8
               movwf   numbits
               clrw
               call    outbits

               ; F10:F7 = 0, F6:F3 set as explained above
               movlw   8
               movwf   numbits
               ;movlw  B'00001001'
               movlw   B'00001000'    ;f3--f5--110 active high
               call    outbits

               ; F2:F1 and C2:C1 set as explained above
               movlw   4
               movwf   numbits
               movlw   B'00110000'
               call    outbits

               ; latch the bits into the synthesizer and return
               call    latchbits
               retlw   0

; ******************************
; initR
; Initialize the R register in the synth to divide by 48 (19.2 MHz ->  400kHz)
;
;      R-reg
;     LSB   |----------------  R   ----------|-- TEST ---|MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       0  0 0 0 0 0 1 1 0 0 0  0  0  0  0  0  0  0  0  0  1

initR          ; begin by writing top 5 bits to register
               ; (sets lock-detect precision to one and zeros test mode bits)
               movlw   5
               movwf   numbits
               movlw   H'80'
               call    outbits

               ; next, write R14:R7 divisor bits as all zeros
               movlw   8
               movwf   numbits
               movlw    b'00000000'
               call    outbits

               ; write R6:R1 = 08h to give divide by 8
               movlw   6
               movwf   numbits
               movlw    b'11000000'
               call    outbits

               ; finally, write two lsb's as zero to designate R register
               movlw   2
               movwf   numbits
               clrw
               call    outbits

               ; latch the bits into the synth and return
               call    latchbits
               retlw   0

; ***********************
; inc_chan
```

```
; Increments the current channel, wrapping to zero if max channel reached
; ************************

inc_chan       incf    channel, F
               movlw   NUMCHAN + 1
               subwf   channel, W
               btfsc   STATUS, Z

               clrf    channel
               retlw   0


; *********************************
; writeN
; N = 2255 --> B=70  A=15
; 902e6 = [32*B + A] * (19.2e6/48)
;
;      N-reg :
;     LSB   |--- A ---|---------------  B ---------------| MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       1  0 1 1 1 0 0 0 1 1 0  0  0  1  0  0  0  0  0  0  0
writeN
               ;msb
               movlw  1
               movwf  numbits
               clrw
               call   outbits

               ; write first 8 bits of B N18:N11
               movlw  8
               movwf  numbits
               movlw   b'00000010'
               call   outbits

               ; write last 5 bits of B  N10:N6
               movlw  5
               movwf  numbits
               movlw   b'00110000'
               call   outbits

               ; write last 5 bits of A  N5:N1
               movlw  5
               movwf  numbits
               movlw   b'00111000'
               call   outbits

               ; write two control bits designating this as N register
               movlw  2
               movwf  numbits
               movlw  H'40'
               call   outbits

               ; latch into synth and return
               call   latchbits
               retlw  0

; ***********************************
; wait250ms
; Delays by about 250 milliseconds
; ***********************************

wait250ms      movlw  250
               movwf  count          ; outer loop ms counter

wait250ms_outer        movlw  250
               movwf  temp           ; inner loop counter (250 times 4us = 1ms)

wait250ms_inner        decf   temp,F         ; dec inner counter -- 1us
               btfss  STATUS,Z       ; check zero flag  -- 1us
               goto   wait250ms_inner ; continue if not zero -- 2us
```

```
                decfsz  count, F       ; dec outer counter
                goto    wait250ms_outer        ; continue if not zero

                retlw   0


; ************************************
; The main routine
; ************************************

start           ; initialize the PIC and its I/O ports
                call    initGPIO

                ; initialize the synthesizer and channel number
                call    initF          ; initialize the programming setup regs
                call    initR          ; initialize the ref freq divisor
                call    writeN         ; write the N divisor for this channel

                ; clear output (especially for when the last bit is '1')
                bcf     GPIO, DATABIT
                bcf     GPIO, CLKBIT
                bcf     GPIO, LATCHBIT

                bsf             GPIO,LED1
                call    wait250ms
                call    wait250ms
                bcf      GPIO,LED1

main_loop
; loop, checking channel switch for press (SWITCHBIT = 0)

                btfsc   GPIO, SWITCHBIT
                goto     main_loop

                call    writeN         ; write the N divisor for this channel

                bcf     GPIO, DATABIT
                bcf     GPIO, CLKBIT
                bcf     GPIO, LATCHBIT

                ; clear output (especially for when the last bit is '1')
                bsf             GPIO,LED1
                call    wait250ms
                bcf      GPIO,LED1

                goto main_loop

        end
```

F.4 2400MHz Board

```
; Frequency synthesizer controller for EECE662/664 GOES Satellite Downconverter
; By           William B. Kuhn
; Created      11/25/00
; Revised      12/01/06 for use with 12F509
; Revised   8/17/07 for downconversion of LRIT signal to 138.000 MHz

; Modified  Xiaohu Zhang
; Modified to be used for 2.4GHz transmiter
;      f_vco = 2.4e9 = [32*B + A] * (19.2e6/R)  --> 2.4e9 = (N*19.2e6)/R
;      when N = 1000 then R = 8
;          thus B = 31, A = 8 and R = 8
;
;      N-reg :
;     LSB    |-- A --|---------------  B ---------------| MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       1  0 0 0 0 1 0 1 1 1 1  1  0  0  0  0  0  0  0  0
;
;      R-reg
;     LSB    |---------------  R   -----------|-- TEST --|MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       0  0 0 0 0 1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  1


; This program sets up the National Semiconductor LMX2326 synthesizer chip
; and increments through channels when a switch is pressed.
; The synthesizer is programmed for a reference frequency of 500 kHz (1/8 of 4 MHz
; clock), and programs one of 10 channels from 138 MHz to 147 MHz.
; On powerup, the channel is set to 138 MHz.  Each press of the channel up
; switch will advance to the next channel, 1 MHz higher.  After channel 10
; is reached (147 MHz), the synth is programmed to an invalid channel to force
; the lock light to go out (useful for testing and also for knowing where one
; is in the sequence).  At the next press, channel 1 is again programmed
; and the sequence repeats.



; Meta data for the compiler
; This is for a 12f509 with 1024 words of program and 41 file registers
; However, for ease of use (no bank switching), we will limit code to 512 words and 25 file regs
       list    p = 12f509
       radix   dec
       include <p12f509.inc>

; Configure the chip for MCLR disabled, Code protect off,
; WDT disabled, and crystal oscillator
       __config       H'0a'


; Symbolic constants

NUMCHAN        equ     20      ; number of channels we can tune to
CLKBIT         equ     0       ; GPIO bit 0 is clock bit on synth
DATABIT        equ     1       ; GPIO bit 1 is data bit on synth
LATCHBIT       equ     2       ; GPIO bit 2 is LE (latch) bit on synth
SWITCHBIT      equ     3       ; GPIO bit 3 is channel up switch

LED1           equ 5   ; GPIO bit 5 is LED1

; Register assignments

channel        equ     8       ; current channel
numbits        equ     9       ; # of bits to write to synth
Ndiv           equ     10      ; low order byte of N reg
temp           equ     11      ; temporary
count          equ     12      ; counter for delay loop
```

```
; ******************  Start of program *************************

; Setup reset vector to skip over subroutines, which must reside
; in first 256 addresses
       org     0
       goto    start

; *************************
; initGPIO
; Initialize I/O port direction and set bits to zero
; *************************

initGPIO       ; first set the option register bits to allow GP2 to be an output
               movlw   B'10001111'
               option

               ; next, set the tristate register to enable GP0 -> GP2 and GP5 as outputs
               movlw   B'11011000'
               tris    6

               ; clear the outputs (probably redundant with reset state, but hey...)
               bcf     GPIO, DATABIT
               bcf     GPIO, CLKBIT
               bcf     GPIO, LATCHBIT
               ;bcf GPIO, SWITCHBIT

               retlw   0

; ************************
; outbits
; Shifts number of bits specified in 'numbits' from W into synthesizer.
; Data is output msb first (left shifted)
; ************************

outbits        movwf   temp    ; save data passed in from W

outbits_loop   ; branch to set or clear bit depending on value of msb
               btfsc   temp,7
               goto    outbits_set
               goto    outbits_clr

outbits_set    ; set or clear the data bit on the synth
               bsf     GPIO, DATABIT
               goto    outbits_clk
outbits_clr    bcf     GPIO, DATABIT

outbits_clk    ; toggle the clock line to input bit to synth
               bsf     GPIO, CLKBIT
               bcf     GPIO, CLKBIT

               ; process the next bit, or return if all done
               rlf     temp, F
               decfsz  numbits, F
               goto    outbits_loop
               retlw   0

; ************************
; latchbits
; toggles the LE line on the synth to latch the bits shifted in with outbits
; ************************

latchbits      bsf     GPIO, LATCHBIT
               bcf     GPIO, LATCHBIT
               retlw   0


; *********************************
; initF
; Initialize the F register (programming reg) on the synth.
; VCO slope as + (F6=1), digital lock detect is selected (F5:F3=001),
```

```
; Powerdown/reset is disabled (F2:F1 = 00), and the initialization mode is
; selected (C2:C1 = 11).
; *********************************

initF
               ; F19
               movlw  1
               movwf  numbits
               clrw
               call   outbits

               ; F18:F11 = 0
               movlw   8
               movwf   numbits
               clrw
               call    outbits

               ; F10:F7 = 0, F6:F3 set as explained above
               movlw   8
               movwf   numbits
               ;movlw  B'00001001'
               movlw   B'00001000'    ;f3--f5--110 active high
               call    outbits

               ; F2:F1 and C2:C1 set as explained above
               movlw   4
               movwf   numbits
               movlw   B'00110000'
               call    outbits

               ; latch the bits into the synthesizer and return
               call    latchbits
               retlw   0


; ******************************
; initR
; Initialize the R register in the synth to divide by 64 (19.2 MHz ->  300kHz)
;
;      R-reg
;     LSB    |---------------  R   -----------|-- TEST --|MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       0  0 0 0 0 0 0 0 1 0 0  0  0  0  0  0  0  0  0  0  1

initR          ; begin by writing top 5 bits to register
               ; (sets lock-detect precision to one and zeros test mode bits)
               movlw   5
               movwf   numbits
               movlw   H'80'
               call    outbits

               ; next, write R14:R7 divisor bits as all zeros
               movlw   8
               movwf   numbits
               movlw    b'00000001'
               call    outbits

               ; write R6:R1 = 08h to give divide by 8
               movlw   6
               movwf   numbits
               movlw    b'00000000'
               call    outbits

               ; finally, write two lsb's as zero to designate R register
               movlw   2
               movwf   numbits
               clrw
               call    outbits

               ; latch the bits into the synth and return
               call    latchbits
```

```
                retlw   0

; ************************
; inc_chan
; Increments the current channel, wrapping to zero if max channel reached
; ************************

inc_chan        incf    channel, F
                movlw   NUMCHAN + 1
                subwf   channel, W
                btfsc   STATUS, Z

                clrf    channel
                retlw   0


; *********************************
; writeN
; N = 8000 --> B=250  A=0
; 2.4e9 = [32*B + A] * (19.2e6/64)
;       N-reg :
;     LSB    |--- A --|---------------  B ---------------| MSB
;      C1 C2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
;       1  0 0 0 0 0 0 0 1 0 1  1  1  1  1  0  0  0  0  0  0
writeN
                ;msb
                movlw  1
                movwf  numbits
                clrw
                call   outbits

                ; write first 8 bits of B N18:N11
                movlw   8
                movwf   numbits
                movlw    b'00000111'
                call    outbits

                ; write last 5 bits of B  N10:N6
                movlw   5
                movwf   numbits
                movlw    b'11010000'
                call    outbits

                ; write last 5 bits of A  N5:N1
                movlw   5
                movwf   numbits
                movlw    b'00000000'
                call    outbits

                ; write two control bits designating this as N register
                movlw   2
                movwf   numbits
                movlw   H'40'
                call    outbits

                ; latch into synth and return
                call    latchbits
                retlw   0

; ***********************************
; wait250ms
; Delays by about 250 milliseconds
; ***********************************

wait250ms       movlw   250
                movwf   count           ; outer loop ms counter

wait250ms_outer         movlw   250
                movwf   temp            ; inner loop counter (250 times 4us = 1ms)

wait250ms_inner         decf    temp,F          ; dec inner counter -- 1us
```

```
                btfss   STATUS,Z       ; check zero flag  -- 1us
                goto    wait250ms_inner ; continue if not zero -- 2us

                decfsz  count, F       ; dec outer counter
                goto    wait250ms_outer        ; continue if not zero

                retlw   0


; ************************************
; The main routine
; ************************************

start           ; initialize the PIC and its I/O ports
                call    initGPIO

                ; initialize the synthesizer and channel number
                call    initF          ; initialize the programming setup regs
                call    initR          ; initialize the ref freq divisor
                call    writeN         ; write the N divisor for this channel

                ; clear output (especially for when the last bit is '1')
                bcf     GPIO, DATABIT
                bcf     GPIO, CLKBIT
                bcf     GPIO, LATCHBIT

                bsf             GPIO,LED1
                call    wait250ms
                call    wait250ms
                bcf      GPIO,LED1

main_loop
; loop, checking channel switch for press (SWITCHBIT = 0)


                btfsc   GPIO, SWITCHBIT
                goto     main_loop


                call    writeN         ; write the N divisor for this channel

                bcf     GPIO, DATABIT
                bcf     GPIO, CLKBIT
                bcf     GPIO, LATCHBIT

                ; clear output (especially for when the last bit is '1')
                bsf             GPIO,LED1
                call    wait250ms
                bcf      GPIO,LED1

                goto main_loop

        end
```