DESIGNING, FABRICATING AND TESTING CONCURRENTLY ACTIVE WIRELESS SENSORS

by

David W. Sammel, Jr.

BS, University of Pittsburgh, 2003

Submitted to the Graduate Faculty of

The School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2005

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This thesis was presented

by

David W. Sammel, Jr.

It was defended on

April 5, 2005

and approved by

Ronald G. Hoelzeman, Associate Professor, Department of Electrical and Computer Engineering

Raymond R. Hoare, Assistant Professor, Department of Electrical and Computer Engineering

Thesis Co-Advisor: Marlin H. Mickle, Nickolas A. DeCecco Professor, Department of Electrical and Computer Engineering

Thesis Co-Advisor: James T. Cain, Professor, Department of Electrical and Computer Engineering

DESIGNING, FABRICATING AND TESTING CONCURRENTLY ACTIVE WIRELESS SENSORS

David W. Sammel, Jr., MS

University of Pittsburgh, 2005

There are many possible uses for remotely powered environmental sensing devices. The University of Pittsburgh has obtained a subcontract to assist in the first development phase of one such device for NASA, a wireless temperature sensor that could ultimately be used to measure the temperature of panels on their spacecraft. This thesis describes the work that has been done to completely meet the project specifications set forth in the subcontract, with particular emphasis being given to the contributions made by the author. In addition to the remote sensor board hardware and software, an embedded protocol is developed that can allow hundreds of these devices to transmit their temperature readings over a single communications channel (amplitude shift keying at 418 MHz) without interference or the need for an on-board receiver. Laboratory testing results that verify the proper operation of the final prototype are included.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

I would like to extend a special thanks to Dr. Marlin H. Mickle and Dr. James T. Cain for their valuable guidance during my thesis work and educational career at the University of Pittsburgh. Also, thanks to Dr. Minhong Mi for the generous assistance he provided when I was beginning my work.

1.0 INTRODUCTION

Electronic devices have long been used to monitor environmental conditions. Traditionally, many of these devices, such as video cameras, microphones and thermostats, have been physically wired to their power supplies or equipped with batteries. What if certain environmental sensors could be powered wirelessly from a radio frequency (RF) source? While having an RF-powered camcorder might not be very useful, certain sensing devices and applications could benefit tremendously by employing energy harvesting technology like that developed at the University of Pittsburgh [1].

Wireless sensing technology could be particularly beneficial when large numbers of sensors are to be used together. Perhaps many devices are spread over a physical area to obtain a spatial profile of some environmental quantity. Considering that a large number of devices are in use, it would not be best to hardwire their power connections. This could lead to a great amount of wiring (hence increased cost and construction time) and the possibility of a cable malfunction disabling one or more sensors. Using batteries is not desirable because they tend to be very large physically compared to the size of modern electronics. Therefore, a battery could be the feature that limits the minimum size of each sensor. Furthermore, changing the batteries could be difficult or impossible if the sensors were to be embedded inside an area or material. Powering the sensors wirelessly would be the most flexible implementation. The remote devices could be placed anywhere within range of the RF energy source (which could be mobile itself) and could even be permanently embedded within a structure.

The sensor network would be even more robust if the sensor devices were to return their environmental readings via a wireless medium as well. Again, this eliminates the chance of a wire failure causing readings to be lost, and reduces the physical complexity of the network as a whole. A very small, completely wireless sensor device would have the flexibility to be placed practically anywhere so long as RF signals could be transmitted to and from it.

One issue that arises when networking this type of device is that of access control. Consider that two dedicated frequency channels are available, one for powering the sensor devices and one for the transmission of the environmental readings (the data channel). When several devices are powered simultaneously by the RF energy source, how will they synchronize their outgoing data transmissions such that they do not interfere with one another? One solution is to equip each device with a receiver for the data channel and use a carrier-detect scheme to determine when it is free. The drawback to using this approach is that the receiver circuitry will consume space and power on the device. An alternative solution to this problem is to assign each device a unique time slot during which it must transmit. This eliminates the interference concern as long as each device maintains an accurate count of the current time slot number. In addition to removing the need for a data channel receiver, this simple access protocol would also have a very straightforward software implementation. However, a method must still be devised to synchronize the start of each time slot across multiple sensors.

This thesis describes my contributions to the development of a prototype for a completely wireless temperature sensing device. NASA identified the need for wireless temperature sensors that could be used to measure the temperature of panels on their spacecraft. These sensors would need to be powered by a remote energy source and also transmit their temperature readings

wirelessly. Also, a non-interference protocol like those described above must be used to prevent multiple sensors from simultaneously transmitting on the data channel.

Gnostic Communications was one of the participating groups in the first development phase of this device. A subcontract to perform part of the Phase I development was given to Dr. Marlin H. Mickle and Dr. James T. Cain at the University of Pittsburgh. My responsibility was the design of the testing and optimization methodology and analysis for optimization of the power consumption and communication protocols of the remote temperature sensing device.

2.0 PROBLEM STATEMENT

The goal of my research was to optimize the energy harvesting, power consumption, communication protocols and the mutual interactions of the NASA specified temperature sensing device. Through my research, I developed a fully operational device to satisfy the requirements. The design and development criteria are presented in this chapter.

The first criterion dealt with physical aspects of the sensor board. Existing remote sensor devices developed at the University had a form factor of 1.7" by 3.2". The final device for NASA must have a form factor of at most 1" by 2". Although I was not responsible for making the initial reduction in form factor to meet this requirement, it was necessary to maintain a minimal board size when making various updates to the final device PCB layout. The specifications were that surface mount integrated circuits and discrete components must be used on the board instead of the "through the hole" type components used in previously developed devices.

An embedded non-interference protocol must be developed and fully implemented in software. This protocol must be easily extendable to allow communication with hundreds of sensors even though a small set (10) will be used in this development phase. No specification with respect to approach was given, but the possibility exists of assigning each device a unique identification number and using delays for time division multiplexing based on these identification numbers to create a time-slotted protocol. This approach will be used. A different

mechanism had been originally developed by Minhong Mi that can be further refined to be employed in the final device.

At least 10 sensor boards must be demonstrated in a laboratory setting to test the non-interference protocol while reporting (communicating) valid temperature readings. These devices must be read (that is, they must all obtain and transmit temperature readings) in one tenth of a second while adhering to the inherent protocol. This is the primary constraint on the protocol design to guarantee that multiple boards do not transmit simultaneously.

The minimum Baud rate for data channel transmissions is specified as 2,400 bits per second. Each device will send a maximum of a 24-bit frame consisting of an identification number and temperature reading. An analysis will be conducted to determine the amount of overhead necessary for powering and synchronizing the sensor devices.

Multiple prototypes will need to be developed to test and meet all specifications. Successive prototypes should improve upon the previous ones and be thoroughly tested to evaluate techniques or components to be used in the final system. The remaining chapters in this document report the research and results for the major prototypes with particular emphasis on my contributions.

3.0 PROTOTYPES ZERO AND ONE

Two older prototypes (termed zero and one) for this device were available at the start of my research. Background information on prototypes zero and one is presented here to provide a basis for how the subsequent prototypes evolved.

Prototype zero was an older active remote sensor (ARS) device. This first prototype performed the most basic functionality stated in the project specifications. Its primary purpose was to obtain a temperature reading and transmit it to a 418 MHz receiver, which is the core behavior required in the final prototype. However, this device functioned only as a single device – multiple boards could not be powered simultaneously because an embedded non-interference protocol was not employed. Prototype zero was implemented on a 1.7" by 3.2" printed circuit board and contained three separate integrated circuits – a 418 MHz transmitter, microcontroller and analog to digital converter used to sample the voltage across a thermistor [2]. A fabricated prototype zero board is shown in Figure 3.1.

Prototype one improved on device zero by combining these three integrated circuits into a single package. The Microchip rfPIC12F675K microcontroller contains a 418 MHz transmitter and A/D converter and thus was an ideal choice for use on this prototype. Combining this circuitry allowed for a significant reduction in PCB size. Prototype one met the project size specifications of a 1" by 2" device form factor (Figure 3.2). The problem with prototype one was that it operated only within a few inches of the 915 MHz energy source. This was due to increased power consumption of the single chip compared to prototype zero [2].

With these two devices, the groundwork was laid for prototype two. Prototype two will explore a simple non-interference protocol that could be used to allow multiple simultaneously powered devices to transmit data without interference. It would also be used to examine the power consumption of another combined microcontroller/transmitter device, the rfPIC12C509AG.

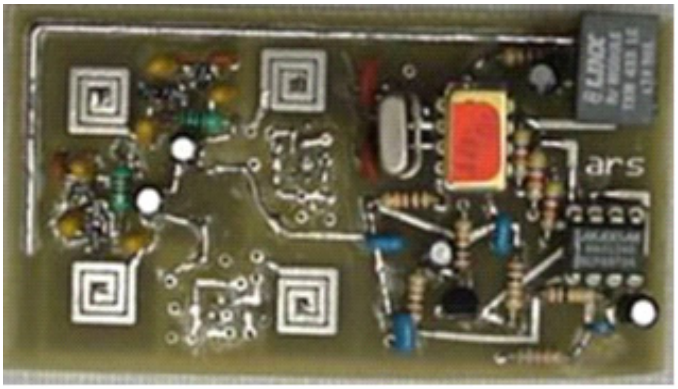


Figure 3.1 Fabricated Prototype Zero Board

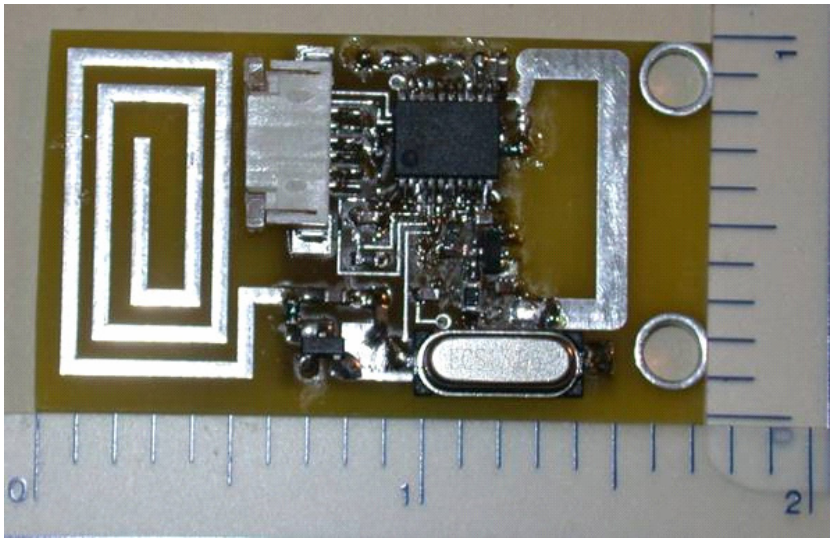


Figure 3.2 Fabricated Prototype One Board with Dimensions Indicated

4.0 PROTOTYPE TWO

The purpose of the development of prototype two was twofold. First, the prototype two devices employed a non-interference protocol as required by the project specifications. The successful completion of prototype two would provide a "proof of concept" for a simple protocol. Second, the design, fabrication and testing of these devices provided an invaluable opportunity to analyze the device operation that proved to be necessary when developing future prototypes.

The requirements for this prototype were very straightforward. The device must be capable of transmitting a 96-bit identification number when power is applied. Although the value of the 96-bit number is not significant here, it could practically represent a variety of useful data, including temperature readings and bar codes. The method of modulation is amplitude shift keying at a carrier frequency of 418 MHz. A simple non-interference protocol must be used to allow several of these devices, when powered simultaneously, to transmit their identification numbers without interfering with one another. The prototype two devices were to be powered initially by wire connections to a standard supply. Although low-power design was not a primary consideration for this prototype, a power analysis could then be performed after the devices were fabricated to determine if wireless operation is possible.

The prototype two devices do not have 418 MHz receivers that would allow for a carrier sensing non-interference protocol. Therefore, another method had been used to ensure a free channel. The simple protocol used for this prototype is as follows. The amount of time Δt

required to transmit the 96-bit tag ID is determined, and each tag is programmed with a unique "slot number" *s*, where *s* is an integer greater than or equal to 0. Each tag waits for $s(\Delta t)$ time before transmitting, which prevents two tags from transmitting simultaneously. While this protocol is certainly effective, it does require that all devices be powered simultaneously so that each of the $s(\Delta t)$ delays begin at roughly the same time. The possibility for interference exists if any devices power up slightly earlier or later than the rest. This suggests that the protocol might not be suited for a wireless application where multiple distributed devices are powered by a single base station. The varying distances of these devices from the base station could lead to closer devices being powered more quickly than distant ones, hence violating the simultaneous powering requirement of the protocol.

4.1 HARDWARE

The hardware design for prototype two was based on those of the previous prototypes. The circuit schematic is shown in Figure 4.1. The transmitting antenna design was used without modification. After careful analysis, a different microcontroller was used for this prototype. The Microchip rfPIC12C509AG was chosen for this device because it contains a built-in RF transmitter that uses amplitude shift keying modulation [3]. This simplifies the overall design by removing the need for a separate transmitter IC. Also, a complete suite of tools necessary for PIC software development and programming were readily available.

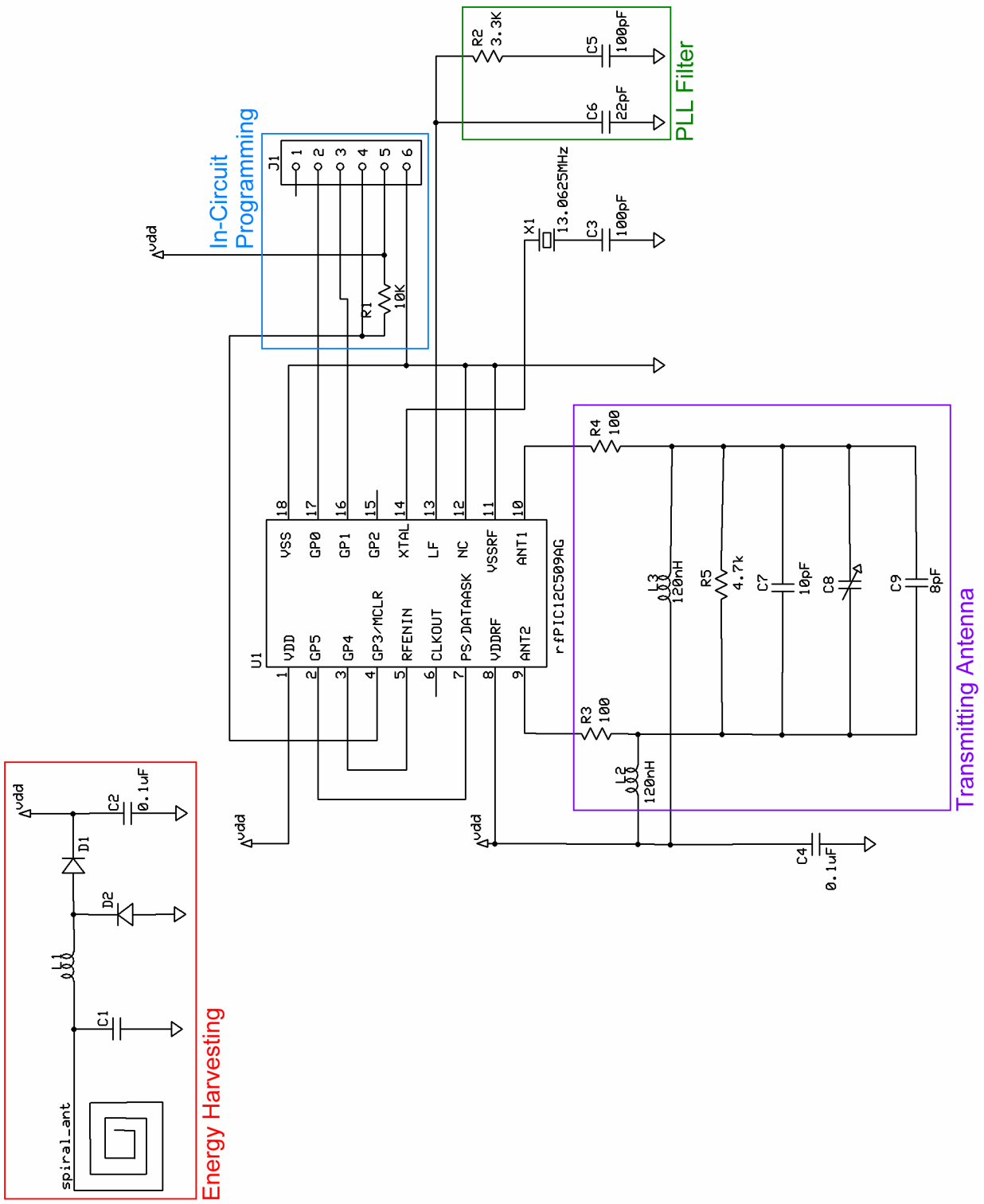


Figure 4.1 Prototype Two Hardware Schematic

For the rfPIC12C509AG, the frequency of the built-in RF transmitter is set by an oscillator external to the microcontroller. To achieve a transmit frequency of 418 MHz, the frequency of the oscillator was selected to be 418 MHz / 32 = 13.0625 MHz as specified in the microcontroller datasheet. The crystal used for this project was manufactured by ECS, Inc. International (part number ECS-130.625-CD-0373) [4].

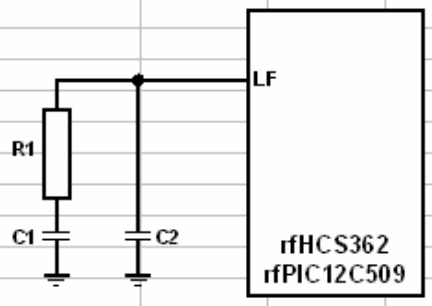
A phase-locked loop (PLL) filter is also required for the internal RF transmitter. The filter design is specified in the microcontroller datasheet, but the end user must choose the component values. Microchip Application Note AN846, "Basic PLL Filters for the rfPIC™ / rfHCS [5]," describes the functionality of the loop filter in detail and explains how to choose appropriate component values using an accompanying Microsoft Excel spreadsheet. The application note states that, "For most designs, the three loop filter components can be quickly found with the spreadsheet calculator [5]." The top portion of Figure 4.2 shows the ideal filter calculations provided by the spreadsheet for a transmit frequency of 418 MHz, including the best values for C1, C2 and R1 (125 pF, 17 pF, 3508 Ω). These values were not readily available, so similar components were chosen for the loop filter (100 pF, 22 pF, 3300 Ω). The bottom portion lists the actual filter parameters given these modified component values. It can be seen that the actual parameters are very close to the ideal calculations, and the actual phase margin is right around the desired value of 50 degrees. The actual loop bandwidth parameter is more important in designs requiring FCC approval because it impacts the amount of RF noise that is transmitted outside of the carrier frequency (418 MHz). For this prototype, remaining close to the default bandwidth of 1 MHz is acceptable. Therefore, the component values chosen are satisfactory for the PLL filter.

AN846 Filter Design Calculator

Rev 1

Step 1: Enter transmitter frequency

Desired System:			**Notes:**	
Transmitter Frequency	418	MHz	Determined by regulations	
Design Parameters:				
Crystal Frequency	13.0625	MHz		
Desired Phase Margin	50	degrees		
Desired Loop Bandwidth	1000	kHz	Reduce loop bandwidth for smaller spurs	
			Increase loop bandwidth for less phase noise	
Device Parameters:				
Multiplication Factor (N)	32	times		
Phase Detector Gain (Kϕ)	0.26	mA		
VCO Gain (Kvco)	250	MHz/V		
LF pin stray capacitance	2	pF		
Filter Calculations:				
Natural Frequency	642509	Hz		
Loop Bandwidth	6283185	rad/s		
Calculated Pole	17262910	Hz		
Calculated Zero	2286892	Hz		
Lock Time	5	us		
Calculated Ctotal	141	pF		
Calculated C1	125	pF		
Calculated C2	17	pF		
Calculated R1	3508	ohms		



Step 2: Enter Actual Components and Tolerances

Actual Values Used:			**Tolerance:**	
Actual Value C1	100	pF	± 5.0%	Recalculate
Actual Value C2	22	pF	± 5.0%	
Actual Value R1	3300	ohms	± 1.0%	

Filter Analysis:	**Min:**	**Typical:**	**Max:**	
Natural Frequency	700015	717302	735936	Hz
Actual Pole	14810815	15656566	16588441	Hz
Actual Zero	2857429	3030303	3222013	Hz
Actual Loop Bandwidth	898622	913462	929076	Hz
Actual Phase Margin	48	50	52	degrees
Lock Time	4	4	4	us

Figure 4.2 PLL Filter Spreadsheet (Included with Microchip AN846)

The energy harvesting and in-circuit programming circuits present in the schematic were taken directly from the existing prototype design. Again, energy harvesting (wireless operation) was not intended to be used initially, but the circuitry was included in the design for future use. It was later found that in-circuit programming would not be needed for prototype two because a UV-erasable DIP version of the rfPIC12C509AG was available.

Figure 4.3 shows the ExpressPCB printed circuit board layout created for prototype two, and a fabricated PCB is depicted in Figure 4.4. Silkscreen component labels on the PCB layout correspond to labels in the hardware schematic. As with the circuit design, the energy harvesting and transmitting antenna portions of the PCB layout were taken from the existing prototype PCB layout. The remainder of the board design was custom made for this prototype to accommodate the rfPIC12C509AG and peripheral circuitry.

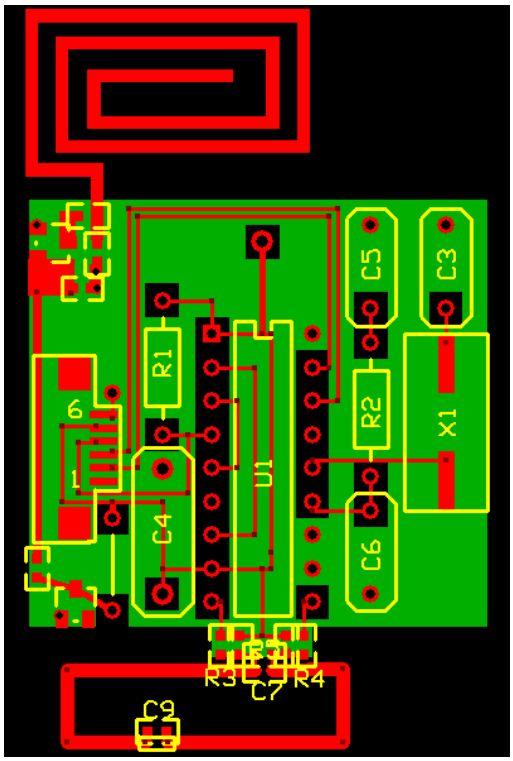


Figure 4.3 Prototype Two PCB Layout

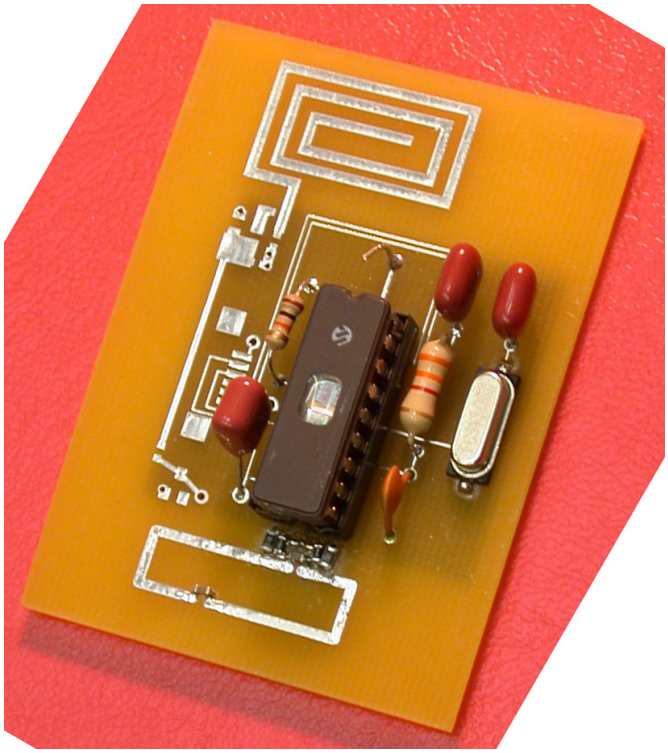


Figure 4.4 Fabricated Prototype Two Board

The energy harvesting and transmitting antenna components on the PCB layout are primarily surface mount components, while several other components are of the through-hole type. The through-hole components were used because they would be easier to solder for this early stage device than surface mount components. Other than the soldering concerns, there is no particular reason for why some components are through-hole and others surface mount. One notable exception is that the UV-erasable rfPIC12C509AG device used was only available in a DIP package. An 18-pin DIP socket was used to allow the PIC chip to be easily swapped on and off the board. This allowed for quicker reprogramming of the microcontroller during software testing than if the IC had to be repeatedly soldered and desoldered.

A power supply may be connected to the circuit by soldering a wire in the via directly above the microcontroller and attaching the positive voltage lead to that wire. The ground lead may be connected to a number of points on the top surface of the board or the ground plane on the bottom side. If the energy harvesting circuitry is to be used to power the board, a jumper wire must be soldered below the programming port on the left side of the board.

4.2 SOFTWARE

The software for prototype two was written in C because it is slightly easier to initially write than assembly language (shorter learning curve) for the new Microchip PICs. The Custom Computer Services, Inc. PCB compiler was used for this purpose [6]. PCB compiler version 3.137 was used, which supports the rfPIC12C509AG (use include file "12C509AG.h" in the compiler "devices" folder). The compiler generates a HEX output file that is used to program the microcontroller. The HEX file may be imported into Microchip's MPLAB IDE, which communicates with a PICSTART Plus programmer to perform the actual programming operation.

To quickly test if a newly fabricated prototype two board is working properly, the following simple C program was devised as a testing program. When the program is executed, a 1 Hz square wave is transmitted by the device. Many parts of the hardware must be functioning correctly for the 1 Hz square wave to be detected at a 418 MHz receiver, including the microcontroller, external oscillator, PLL filter and transmitting antenna. If the test program does not run successfully (i.e., a 1 Hz square wave is not received), then other debugging methods may be used to isolate the source of the problem.

```
#include <12C509AG.h>
#fuses INTRC,NOWDT,NOPROTECT,NOMCLR
#use delay(clock=4000000)

// Internal 4 MHz RC Oscillator Calibration
#rom 1023 = {0xC6C}

main() {

  // Enable RF
  output_high(PIN_B4);

  // Output 1 Hz square wave
  while (TRUE) {
    output_high(PIN_B5);
    delay_ms(500);
    output_low(PIN_B5);
    delay_ms(500);
  }

}
```

Figure 4.5 rfPICTest.c Source Code

First, notice that 12C509AG.h is included, which allows a number of defined constants to be used in the program (e.g., PIN_B4, PIN_B5). The #fuses pre-processor directive specifies various configuration options for the microcontroller. INTRC indicates that the microcontroller's internal 4 MHz RC oscillator is to be used as its clock source. NOWDT and NOPROTECT disable the watchdog timer and code protection features available on the rfPIC12C509AG, respectively. NOMCLR disables the master clear pin (it is tied to the supply voltage internally). This is acceptable because external reset functionality does not need to be supported for this application.

The parameter of the "#use delay" directive is the speed of the processor in Hz (4,000,000 since the internal RC oscillator is being used). This value must be specified if the

delay_ms() function is to be used in the program because it states the relationship between instruction cycles and time.

When using the rfPIC12C509AG internal RC oscillator, the pre-programmed calibration value shipped with the microcontroller should be written to the topmost memory location [3]. The calibration value is erased when the device is erased, so this value must be recorded before the device is erased for the first time and written back to memory location 1,023 each time the device is programmed. The #rom directive accomplishes this. Note that the calibration value 0xC6C is specific to one of the microcontrollers used on a prototype two board; using a different part will require the use of a different calibration value.

Considering the main program, pin 3 (GP4) is tied to the RFENIN pin on the PCB, so driving this pin high enables the RF transmitter in the microcontroller. Pin 2 (GP5) is tied to PS/DATAASK, so toggling the state of this output every half-second results in a 1 Hz square wave being transmitted.

The actual prototype two source code is much longer than that of the test program, so it has been included in Appendix A of this thesis. Please reference this code if necessary while reading the description in the following paragraphs. A pseudocode description of the software is provided in Figure 4.6. This software is used to program each board in order that it waits for its unique time slot and then transmits a 96-bit identification code.

The identification code chosen for each board is, “Board *x* ID”, where *x* is the number of the board. This ID code will ultimately be sent to a PC, where it will be displayed in a terminal window as ASCII characters. To make the display easy to read, after the characters “Board *x* ID” are transmitted, a carriage return and line feed are transmitted as well. Therefore, a total of

```
main() {
        // The board identification number (BOARD) is obtained from a command line
        argument when the software is compiled

        // Setup the 96-bit board identification code
        boardID[13] = "Board   ID";          // A space is left for the identification number
        boardID[6] = BOARD;                  // The ID number is inserted into the ID code
        boardID[10] = Carriage Return;       // CR and LF are for terminal program display
        boardID[11] = Line Feed;             // purposes and to bring the total ID code length to
        boardID[12] = NULL;                  // 96 bits per the prototype two specifications

        Output RF enable signal to turn transmitter on

        // Call delay_ms() function to wait for the correct time slot
        // The length of the delay depends on the unique board identification number (BOARD)
        // which is an integer ≥ 1
        // Each time slot is 56 ms wide
        // Include a 2 ms startup time to make sure that the transmitter on the first board is ready
        // before the first time slot begins
        delay_ms(2 + (56 * (BOARD - 1)));

        // When this point is reached, the delay_ms() function has returned and it is time to
        // transmit the 96-bit board identification code using the RS-232 type format
        // The following for loop iterates once per transmitted ASCII character (see Figure 4.7)
        for(int i = 0; i < 12; i++) {
                Character to transmit is boardID[i]
                Output start bit, 8 data bits and 2 stop bits at 2400 Baud
        }

        Disable RF enable signal to turn transmitter off
}
```

Figure 4.6 Prototype Two Software Pseudocode

twelve 8-bit ASCII characters are transmitted as the identification code, for a total length of 96 data bits. The protocol used for the data transmission is the RS-232 type format, with 8 data bits, no parity bit and two stop bits. The total number of bits that must be transmitted, including classical RS-232 overhead bits, is calculated as:

12 ASCII characters x (1 start bit + 8 data bits + 2 stop bits) / ASCII character

= 12 x 11 = 132 total bits

A Baud rate of 2,400 was chosen for the initial communication protocol because it is one of the specifications for the final prototype. Given this rate and the total number of bits to be transmitted, the length of each time slot Δt may be determined as follows:

$$\Delta t = 132 \text{ bits} \times (1 \text{ second} / 2{,}400 \text{ bits}) = 55 \text{ ms}$$

Considering slight differences in board clock rates, startup times, etc., an additional millisecond of time is arbitrarily added to each slot as extra padding to assure that the end of one transmission does not overlap with the start of the next. Therefore, the slot time used for the prototype two non-interference protocol is 56 ms.

The software listing in Appendix A contains step-by-step comments that explain the functionality of the program. The following is a brief discussion of how the delay times in the RF transmitting *for* loop were calculated (Figure 4.7). At 2,400 Baud, the bit time is 1 / 2,400 = 416⅔ μs. Delays in the rfPIC12C509AG have a resolution of 1 μs, so the bit time is rounded to 417 μs. However, a delay of 417 μs between the transmission of each bit is not correct because some time is used processing instructions between the calls of delay_us(). By analyzing the assembly language code generated by compiling the source code, it was determined that this delay is approximately 8 μs. This explains the calls of delay_us(409) in the RF transmitting loop between each bit transmission. Also, due to the time required for branching from the bottom to the top of the loop (approximately 9 μs), the final delay_us() call has an argument of 408 μs.

```
for(i = 0; i < 12; i++) {
  // Mark to space transition - start bit
  output_high(PIN_B5);
  delay_us(409);   // 417 - 8
  // Output eight data bits, no parity
  // Output low = Mark state = Logic "1"
  // Output high = Space state = Logic "0"
  // Bit 0 – LSB
  if(boardID[i] & 0x01) {
    // Bit 0 is set – output Mark state
    output_low(PIN_B5);
  } else {
    // Bit 0 is cleared – output Space state
    output_high(PIN_B5);
  }
  delay_us(409);
  // Bit 1
  if(boardID[i] & 0x02) {
    output_low(PIN_B5);
  } else {
    output_high(PIN_B5);
  }
  delay_us(409);
  // Bit 2
  if(boardID[i] & 0x04) {
    output_low(PIN_B5);
  } else {
    output_high(PIN_B5);
  }
  delay_us(409);
  // Bit 3
  if(boardID[i] & 0x08) {
    output_low(PIN_B5);
  } else {
    output_high(PIN_B5);
  }
  delay_us(409);
  // Bit 4
  if(boardID[i] & 0x10) {
    output_low(PIN_B5);
  } else {
    output_high(PIN_B5);
  }
  delay_us(409);
  // Bit 5
  if(boardID[i] & 0x20) {
    output_low(PIN_B5);
  } else {
    output_high(PIN_B5);
  }
  delay_us(409);
  // Bit 6
  if(boardID[i] & 0x40) {
    output_low(PIN_B5);
  } else {
    output_high(PIN_B5);
  }
  delay_us(409);
  // Bit 7 – MSB
  if(boardID[i] & 0x80) {
    output_low(PIN_B5);
  } else {
    output_high(PIN_B5);
  }
  delay_us(417);
  // Two stop bits
  output_low(PIN_B5);
  delay_us(417);
  output_low(PIN_B5);
  delay_us(408);
}
```

Figure 4.7 RF Transmitting *for* Loop

4.3 RESULTS

4.3.1 Prototype Two Demonstration

The demo system created to test the prototype two design consists of four boards. Each board receives a unique #BOARD command line parameter (1, 2, 3 or 4) when its software (listed in Appendix A) is compiled. This command line parameter is the value of *x* that will be used for the identification code "Board *x* ID". Viewed another way, the value of *x* is also the number of the board's unique time slot. This value is hard-coded into the microcontroller memory when the software is downloaded.

All four boards are connected to the same power supply as shown in Figure 4.8. When the power supply is turned on, the four tags begin processing their software simultaneously, and each waits for the appropriate time slot before transmitting its ID code. Again, the data are transmitted using amplitude shift keying at a frequency of 418 MHz. A simple 418 MHz receiver accepts the ASK RF signal and converts it to a voltage waveform using standard RS-232 voltage levels. This voltage signal is sent to a PC over a DB9 serial cable, and a terminal program on the computer (ProComm by Datastorm Technologies, Inc.) displays the received ASCII characters on the screen as shown in Figure 4.9. All four ID codes are successfully received.

This result illustrates that the requirements for prototype two have been met. Each device has transmitted a 96-bit ID code using ASK at a carrier frequency of 418 MHz. It can be seen from Figure 4.9 that the non-interference protocol used for prototype two allows the four ID codes to be received without corruption while the devices are powered simultaneously, thereby at least demonstrating a "proof of concept" for this simple protocol.

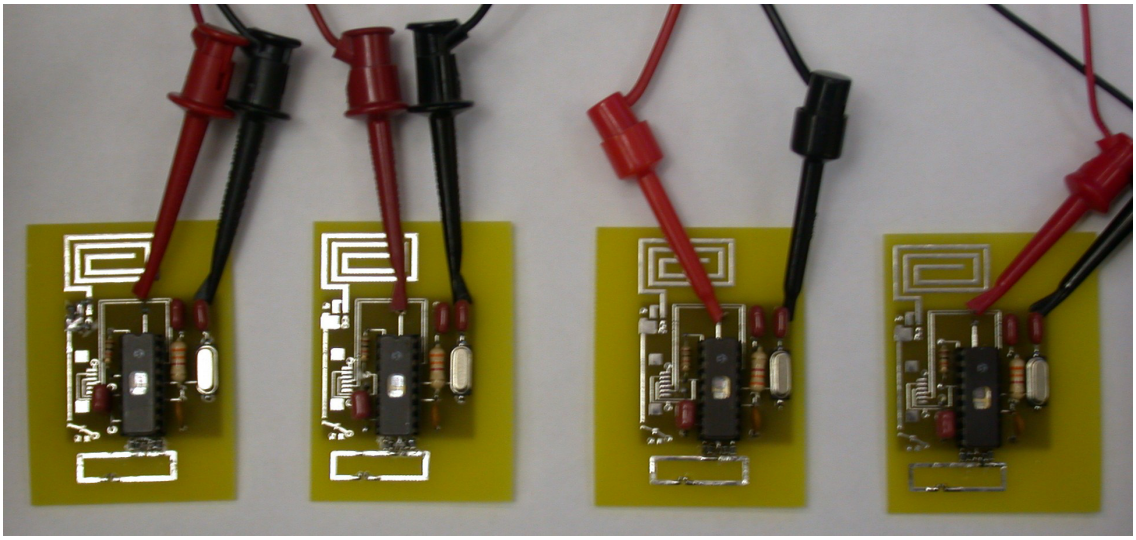


Figure 4.8 Prototype Two Demonstration Setup

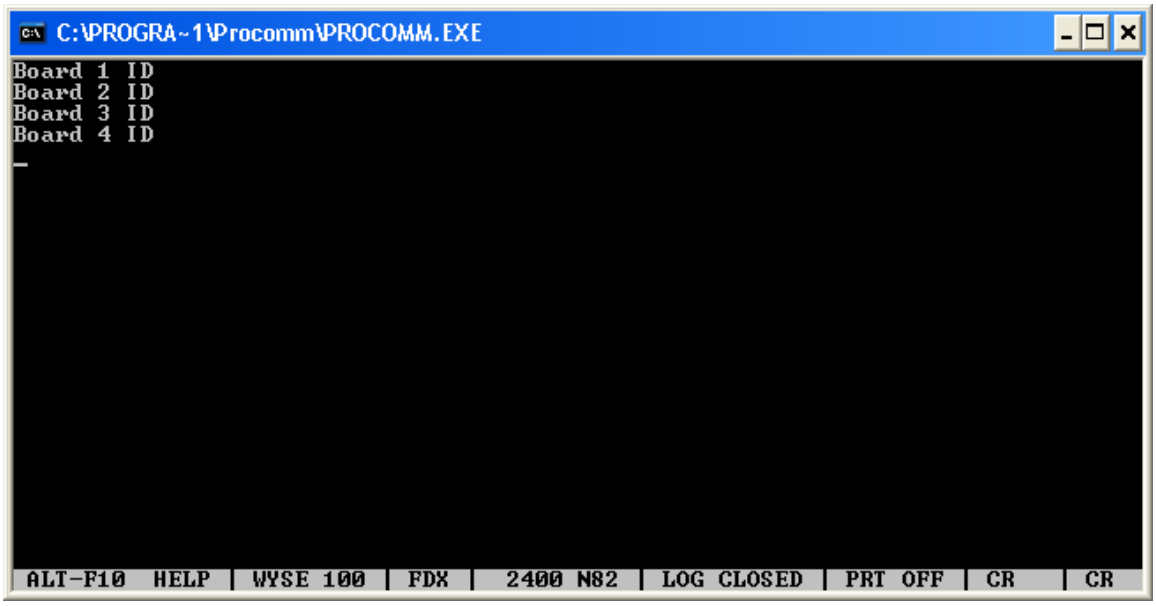


Figure 4.9 ProComm Screen Capture from Prototype Two Demonstration

4.3.2 Power Consumption Data

Although the prototype two boards are "wired" devices, the final prototype requires fully wireless temperature sensing boards. As mentioned previously, the prototype two devices have energy harvesting circuitry onboard that may be used instead of a wired power supply. Using the energy harvesting circuitry was a secondary goal for this prototype. Therefore, it was decided that once a working demonstration had been completed, a power analysis would be performed to determine if energy harvesting would supply enough power to operate the prototype two boards.

A simple C program was used to determine the maximum power consumption of the prototype two boards (Figure 4.10). This program causes the RF transmitter to continuously transmit while an infinite loop prevents the processor from entering a low-power sleep mode.

```
#include <12C509AG.h>
#fuses INTRC,NOWDT,NOPROTECT,NOMCLR
#use delay(clock=4000000)

// Internal 4 MHz RC Oscillator Calibration
#rom 1023 = {0xC6C}

main() {

  int i = 0;

  // Enable RF
  output_high(PIN_B4);
  // Continuously transmit
  output_high(PIN_B5);
  // Prevent PIC from going to sleep
  while(TRUE) {
    i = i + 1;
  }

}
```

Figure 4.10 Power Test Source Code

Referring back to Figure 4.1, note that the schematic shows microcontroller pin 2 (GP5) connected directly to the RF transmitter data input (pin 7, PS/DATAASK). Because the logic "1" output voltage on pin 2 is greater than 2.0V, the transmitter operates at its maximum output power (see Table 7-5 in [3]). The maximum output power was used during device testing, and the four board demonstration, because transmitter power consumption was not a concern for prototype two.

The design would need to be modified and a simple voltage divider circuit added to allow other transmitter output powers to be selected. A board was modified in this way and used to obtain the circuit power consumption measurements presented in this section. Power measurements were made using all six discrete transmitter output powers (see Table 7-5 in [3]). The resulting measurements are provided in Table 4.1. As a baseline, one measurement was also made while the RF transmitter was *disabled* and the processor was not in SLEEP mode.

Table 4.1 Circuit Power Consumption Measurements

Transmitter * Output Power (Datasheet, dBm)	Power Select * Voltage (Datasheet, Volts)	Measured Power Select Voltage (Volts)	Supply Voltage (Volts)	Supply Current (mA)	Circuit Power Consumption (mW)	Transmitter * Operating Current (Datasheet, mA)
+2	≥ 2.0	2.946	3.000	16	48	11.5
-1	1.2	1.267	3.000	16	48	8.6
-1	1.2	1.200	3.000	13	39	8.6
-4	0.9	0.954	3.000	12	36	7.3
-7	0.7	0.778	3.000	11	33	6.2
-10	0.5	0.565	3.000	10	30	5.3
-12	0.3	0.369	3.000	9	27	4.8
-60	< 0.1	0.085	3.000	8	24	< 4.8
Measurements made while continuously transmitting (PIC **not** allowed to sleep).						
RF Disabled			3.000	4	12	

* These values are specified in [3] and were not measured during the power test.

When examining the power measurements, please note that the values in the *Transmitter Output Power*, *Power Select Voltage* and *Transmitter Operating Current* columns were not actually measured during the power test – rather, these values are taken from the microcontroller datasheet.

As shown in Table 4.1, the supply voltage for all of the power measurements is 3.000 V. The power consumption values in Table 4.1 are large enough to indicate that this circuit will not run on power from energy harvesting (approximately 10 mW of power consumption or less would be required). The minimum supply voltages for the microcontroller and RF transmitter are 2.5 V and 2.1 V, respectively [3]. A test was run to get an indication of the circuit power consumption at the minimum supply voltage. Using a supply voltage of 2.500 V and a measured power select voltage of 2.445 V (maximum transmitter output power), 16 mA of current is drawn from the power supply, yielding a circuit power consumption of 40 mW. This value is not drastically different from the power consumption at a 3.000 V supply voltage (48 mW). This leads to the conclusion that it is unlikely that reducing the supply voltage to 2.5 V will allow the board to be powered by energy harvesting.

By examining the transmitter operating current values in Table 4.1, which are taken directly from the rfPIC12C509AG datasheet, it can be seen that the internal transmitter simply seems to draw too much current (several mA) to be powered by the type of energy harvesting circuitry present on the prototype two boards. Therefore, the rfPIC12C509AG will not be a good candidate for the final prototype microcontroller.

4.3.3 Prototype Two Summary

As stated in the introduction to the prototype two section of this thesis, there were two motivations for the development of this prototype. The first was to show that the simple non-interference protocol that was introduced would indeed work, which has been confirmed here by the demonstration results. The second was to gain insight that would be particularly vital for completing the final prototype. This goal was satisfied.

5.0 PROTOTYPE THREE

The development work on prototype three (the final prototype) immediately followed the completion of prototype two. One of the first steps was to set up the existing ARS boards and test them. This would allow for the testing of any software modifications that needed to be made. Significant time was spent examining the hardware schematic to understand in detail how each portion of the circuitry worked. Also, the assembly language code was carefully analyzed to have a firm understanding of the entire program.

With this work completed, the stage was set to begin the work with prototype three.

5.1 HARDWARE

As mentioned previously, the initial hardware design for prototype three was created as an extension of prototypes zero and one. The hardware design had been worked on extensively. However, a number of changes to the PCB layout were needed compared to the latest hardware design.

On prototype three, a voltage divider circuit is used to indirectly measure the resistance of the thermistor and, hence, the temperature. The supply voltage for the voltage divider is output on a microcontroller general purpose I/O pin, which allows the voltage divider circuit to be enabled in software only when a temperature reading is to be performed (this is a low-power consideration). Previously, the microcontroller supply voltage had been used as the A/D

converter reference. This is not the correct choice for the reference voltage because the supply voltage for the temperature sensing circuit is that which is output from the I/O pin, not the microcontroller supply voltage, and these may differ. Therefore, the voltage output from the I/O pin should be used as the A/D converter reference voltage. A specific microcontroller pin is reserved for inputting an A/D converter reference, but it had already been used to control the 418 MHz transmitter in the existing hardware schematic. Because another general purpose I/O pin was free, the transmitter enable signal was connected to the free pin and the voltage divider supply connected to the A/D converter reference input. A software update was written because this change involved modifying the pinout of the microcontroller.

For the PCB modifications, I first identified a pull-up resistor that was present on the hardware schematic but not on the PCB and added it. Also, I had learned from previous testing that connecting an oscilloscope probe to the board in a consistent manner was absolutely necessary for obtaining repeatable voltage measurements. Measurements of the ARS board supply voltage are often important; therefore, I added a through-hole pad that is connected to V_{dd}. A small wire can be soldered here, providing a location at which an oscilloscope probe can be easily and consistently attached for supply voltage measurements. A similar designated location for the oscilloscope ground is not needed because the entire bottom side of the ARS board is a ground plane. Finally, the footprint for a surface mount switch was added that would allow either V_{dd} or ground to be connected to one of the general purpose I/O pins on the microcontroller. I chose to add this capability because it has many possible uses, including enabling/disabling in-circuit programming of the PIC. With the existing printed circuit board design, a small wire needed to be removed from the board whenever reprogramming was necessary. Not only was this inconvenient, but it wasted a significant amount of time when

small software changes needed to be quickly tested. Also, the repeated soldering and desoldering of the wire could eventually cause damage to the board, such as the metal pads lifting off.

The final hardware schematic for prototype three is shown in Figure 5.1, and Figure 5.2 shows the updated printed circuit board layout. When working on the layout, special attention was given to keep the total area of the board to a minimum. This is important because the project specifications state that the total size can be no larger than 1" by 2", and certainly an even smaller size would be desirable. The initial PCB layout had a size of 0.864" by 1.656", and the updated layout of Figure 5.2 is nearly the same size (0.864" by 1.654").

Once the PCB layout had been updated, an order was placed with ExpressPCB for a total of four printed circuit boards, each of which contained four ARS boards. A band saw was used to cut the boards apart and their rough edges were sanded. The next task was to fabricate the boards by soldering the numerous surface mount components onto them by hand. A fabricated prototype three board is shown in Figure 5.3.

Although the SAW resonator component was hand soldered onto the first ARS board, a reflow soldering machine was used to attach these parts to the remaining boards. The resonator package is designed such that reflow soldering is the easiest and most reliable way to obtain a good connection to the board.

The final components that would need to be added to the ARS boards were the inductor and capacitor for the antenna impedance matching.

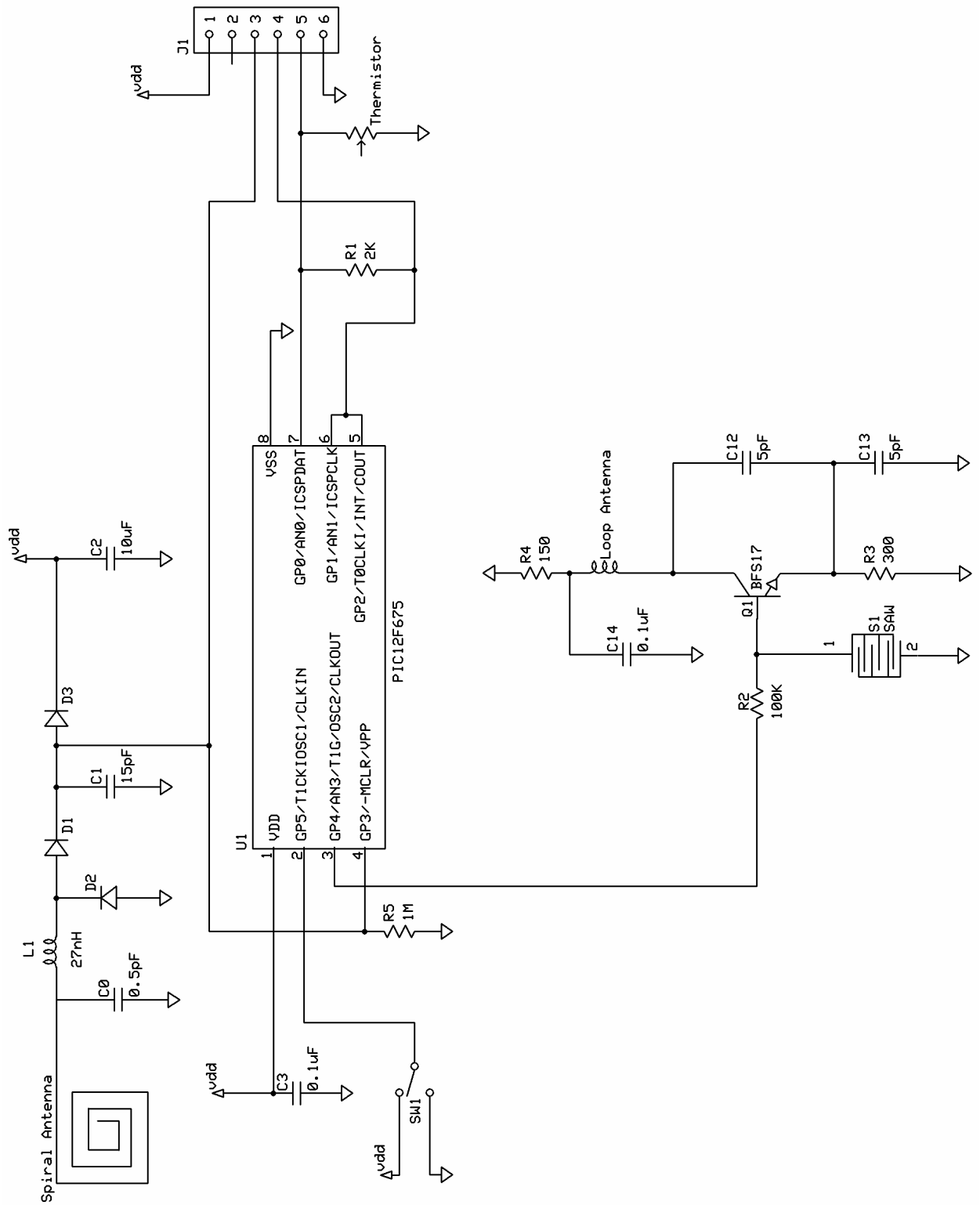


Figure 5.1 Prototype Three Hardware Schematic

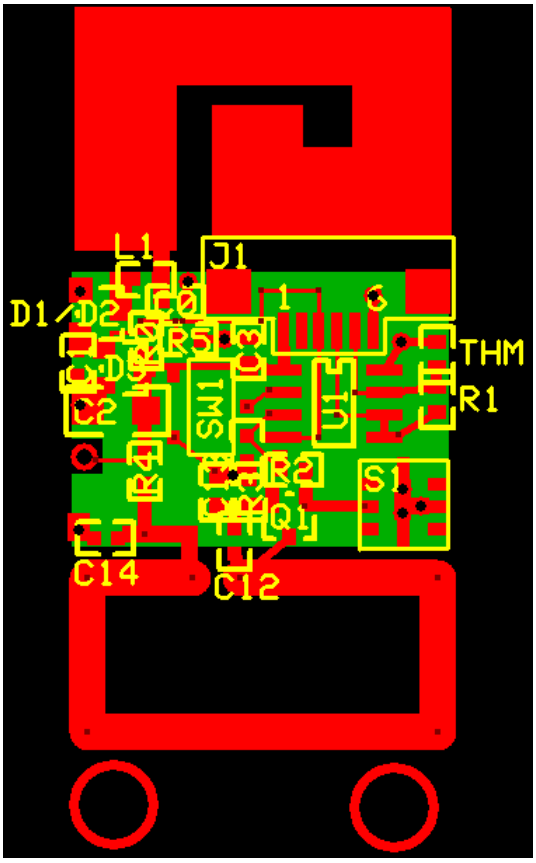


Figure 5.2 Prototype Three PCB Layout

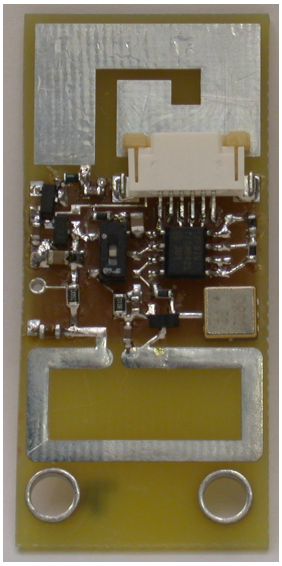


Figure 5.3 Fabricated Prototype Three Board

5.2 ANTENNA IMPEDANCE MATCHING

5.2.1 Initial Impedance Matching Tests

Once the prototype three boards were fabricated, the impedance matching between their antennas and the energy harvesting circuitry was performed. The maximum amount of power may be transferred from the energy harvesting antenna to the storage circuitry by striving to obtain the best impedance matching possible. However, the impedance of the antenna cannot be measured in a straightforward manner due to a number of factors, including the fact that the antenna is physically much smaller than the 915 MHz wavelength (~1 foot) [7]. Therefore, the process described in "Annealing Approach to the Impedance Matching of Antennas for RFID Tags [7]" was used to determine the appropriate impedance matching components.

Before the matching process could begin, some preparation was needed in the lab. First, one of the ARS boards was modified such that two standard header pins protruded from the back of the board. These pins were soldered to the board voltage supply (V_{dd}) and ground, and would be used to connect the ARS board power supply to a separate measurement PCB. This PCB contains an analog-to-digital converter that converts the voltage input to a digital value and sends it over an infrared link to the "Virtual Power Meter" MATLAB software running on a PC. The voltage measurement board and MATLAB software were previously created at the University for this application.

As part of the impedance matching process, it is necessary to estimate the impedance looking into the ARS board just past the antenna (Z_{Rct} in Figure 5.4, which is taken directly from [7]). Minhong Mi had previously developed a small PCB that is used for this purpose. A value for the load resistor R_L, which represents the circuitry on the actual ARS board, is needed to obtain the Z_{Rct} measurements. R_L is chosen such that the resulting current draw is similar to the

current that would be drawn by an actual ARS board during the impedance matching tests. To determine what resistance value should be used for R_L, an estimate of the current consumption on the ARS board is required. The PIC microcontroller datasheet may be used to find this information, as explained in the following paragraph.

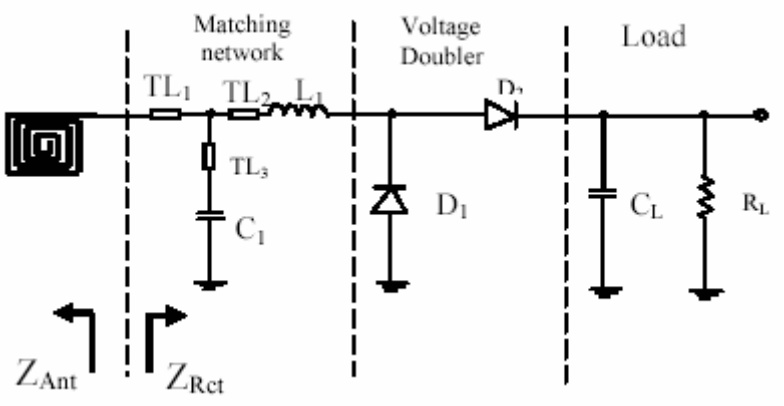


Figure 5.4 ARS Board Equivalent Circuitry

During the matching tests, the PIC on the ARS board simply sleeps. This behavior is appropriate because the microcontroller is typically in a low-power sleep state during normal ARS board operation and it allows for an accurate estimation of R_L due to the constant current consumption of the PIC during sleep [7]. For the PIC configuration used with the ARS software (watchdog timer, comparators, CV_{REF} and T1OSC disabled, brown-out detect enabled) the current consumption is 70 μA at V_{DD} = 3.0 V and 130 μA at V_{DD} = 5.0 V [8]. The resistances drawing these currents at voltage drops of 3.0 and 5.0 V are 42.9 and 38.5 kΩ, respectively. During the matching tests, V_{DD} will vary dramatically depending on the effectiveness of each match, so it is necessary only to choose a value for R_L that is close to these. A readily available resistor with a measured value of 41.5 kΩ was used.

Referring back to Figure 5.4, the value of R_L has now been chosen. C_L for the ARS board is 15 pF. An RF Network Analyzer can be used to measure the value of Z_{Rct} for a given matching network (values L_1 and C_1). Prior to beginning the matching procedure, the RF Network Analyzer was calibrated so that the impedance measurements would be accurate.

For the impedance matching tests, the RF energy source continually transmits, which is a different behavior than during regular ARS board operation (the non-interference protocol is not in use). As previously mentioned, the PIC sleeps during the test. The tests were performed at a distance of 1.3 meters from the energy source. This distance was chosen experimentally. At closer distances, the voltage readings for poor matches were found to be fairly high in the range of 0 – 5 V (the A/D converter input range), indicating that the voltage readings for better matches might all register as 5 V. This would make it impossible to tell which match is the best. At 1.3 meters, several low readings were observed. Therefore, it seemed that at this distance even the best matches might not generate a supply voltage exceeding 5 V, which would allow a single best match to be identified.

The voltage obtained with the energy harvesting circuitry on the ARS board is converted to a digital value on the measurement PCB and ultimately displayed on a PC. For each test, the value of Z_{Rct} is measured, and the voltage value is plotted versus Z_{Rct} on a Smith chart [7]. Table 5.1 contains the testing results obtained.

The Smith chart in Figure 5.5 visually depicts the testing results. Viewing the results graphically is useful because it helps in identifying trends in the data (i.e., what ranges of LC values lead to better matches). Colors at the red end of the spectrum correspond to higher measured voltages and hence better impedance matches. The best match found for the particular ARS board tested was 27 nH and 0.5 pF, which yielded 3.018 V at a distance of 1.3 meters.

Table 5.1 Impedance Matching Measurements

Matching Network		Z_{rct} Measurement		Voltage Measurement (mV) @ 1.3 meters
Inductance (nH)	Capacitance (pF)	Real Part (Ω)	Imaginary Part (Ω)	
15	0	6.9	-29.5	612.22
18	0	7.8	-11.1	784.22
22	0	8.2	14	1262.33
27	0	9.2	51	2616.30
33	0	15.4	114	637.58
39	0	20.5	175	248.07
12	0.5	5.4	-33	448.47
15	0.5	5.5	-22	551.43
18	0.5	4.2	-65	125.87
22	0.5	9.4	19	1522.31
27	0.5	14.6	65	3017.73
33	0.5	37	168	314.05
39	0.5	300	580	130.45
47	0.5	1600	-300	282.59
15	1	5	-19.8	551.73
18	1	6.4	-5.8	766.80
22	1	9.9	16.1	1683.31
27	1	25	84	967.82
33	1	230	380	331.47
39	1	420	-610	329.02
47	1	58	-300	44.60
15	2	3.4	-14.2	506.82
18	2	5.4	-4	129.84
22	2	10.3	13.4	1420.27
27	2	37	64	1027.70
33	2	45	-168	352.85
39	2	10	-96	176.27
47	2	6.6	-82	28.11

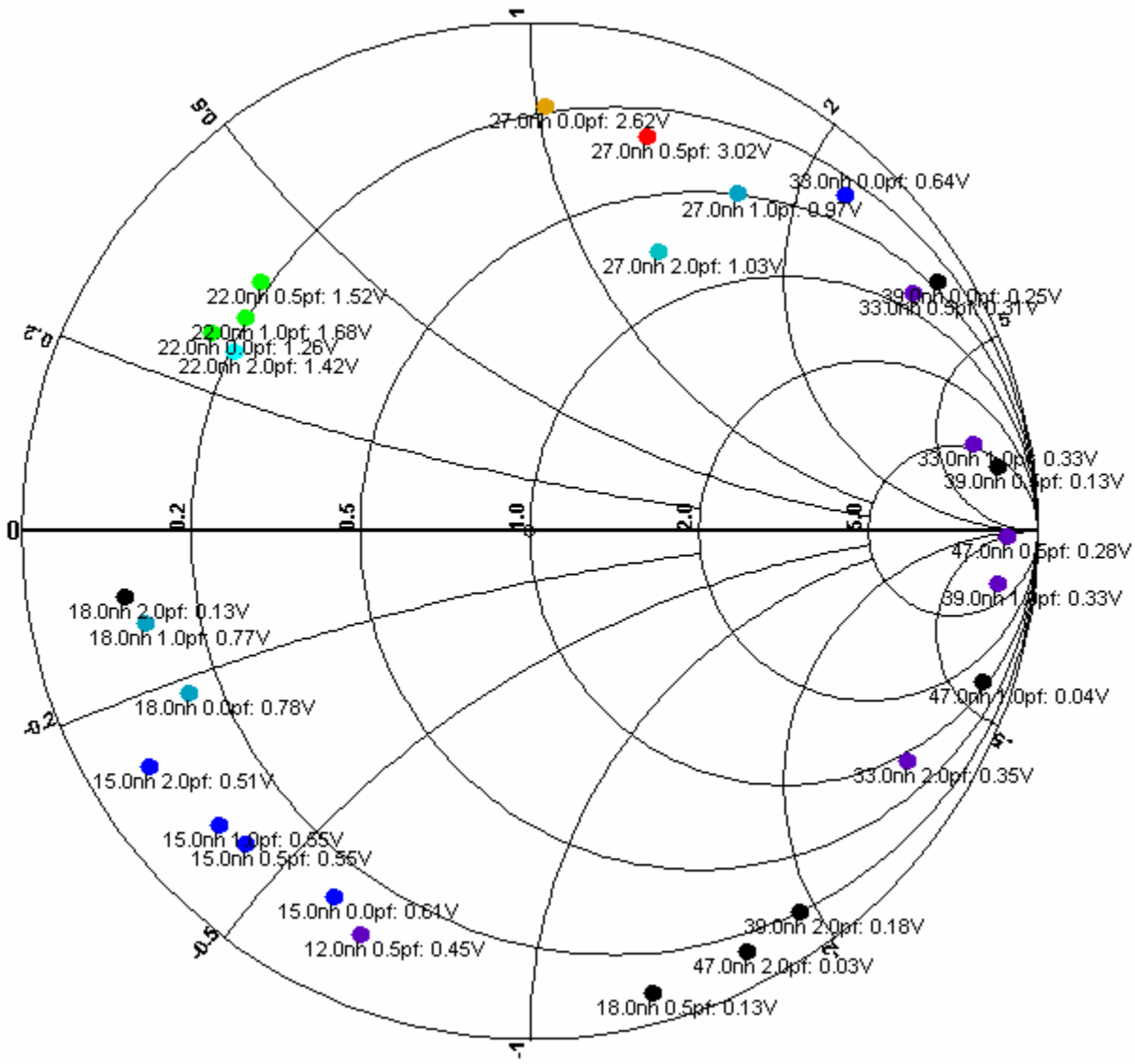


Figure 5.5 Impedance Matching Smith Chart

The impedance matching tests included many standard inductances (15, 18, 22, 27, 33, 39 and 47 nH) and capacitances (0, 0.5, 1 and 2 pF). After testing and plotting all of these combinations, it was seen that the best matches appear with an inductance of 27 nH. This inductance was then tested with some additional capacitance values as shown in Table 5.2. The voltage measurements recorded are the average of two trials.

Table 5.2 Additional Measurements with L_1 = 27 nH

Matching Network		Z_{rct} Measurement		Voltage Measurement (mV) @ 1.3 meters
Inductance (nH)	Capacitance (pF)	Real Part (Ω)	Imaginary Part (Ω)	
27	0	9.4	55.5	2645
27	0.2	10.6	56.4	2725
27	0.4	12	58.6	2745
27	0.5	12.5	59.2	2745
27	0.6	13.6	61	2760
27	0.7	14.4	62	1892.5

From this data, it is seen that capacitance values between 0.2 and 0.6 pF give the best matching with an inductance of 27 nH. Less voltage is obtained when capacitance values above 0.6 pF are used (this agrees with the previously obtained data in Table 5.1). Given that the voltages obtained from 0.2 to 0.6 pF are comparable and a good supply of 0.5 pF capacitors was available, 27 nH and 0.5 pF were initially chosen as the matching values for the prototype three boards. It was later found that slightly different capacitance values provided a better match for some of the remaining nine prototype three boards.

5.2.2 Impedance Matching for the Remaining Boards

Once the initial antenna impedance matching measurements had been made on one ARS board, the next step was to place matching components on the remaining devices and test the quality of the matches. The values of 27 nH and 0.5 pF would not provide the best match for all ten boards because of slight manufacturing differences between the printed circuit boards. Manufacturing differences, particularly involving the thickness of the metal layer, may cause the antenna impedance on other boards to vary slightly from that of the first tested board. This creates the need to check the matching of each board individually and improve matches that are poorer than the one obtained for the first board.

Because only small deviations from the initial matching values were expected and to save time, the infrared voltage measurement board and MATLAB software were no longer used to determine the matching efficiency of the remaining boards. Rather, the maximum operational distance of the each board from the base station was used as an indicator of matching quality. This is acceptable because better impedance matching leads to greater amounts of harvested energy which corresponds to longer maximum operational distances.

The procedure followed for each remaining board is as follows. First, the initial matching values (27 nH and 0.5 pF) are placed on the board and the maximum operational distance is tested. If this distance is comparable to that of the ARS board used in the initial impedance matching process (approximately 70 cm), then the matching quality is considered to be acceptable. This criterion has been established because the matching for the first board was found to be the best after trying many LC combinations (documented in the previous section). However, if the distance is not close to 70 cm, then slightly different matching values are tried, and the process repeats until a good maximum operational distance is found. The previously

performed impedance matching experiments have shown that the best matches occur with an inductance of 27 nH. Therefore, if different matching values need to be tried, the capacitance value will be changed first.

It can be difficult to remember the status of each board when working with ten of them, so I decided to keep a careful record of the status of each board and my tests with them. The record provides a detailed account of the time-consuming impedance matching process followed for each board. It is presented here as a list of status reports, numbered in chronological order.

1. Boards 2 and 5 successfully tested at a distance of 70 cm with the initial matching values of 27 nH and 0.5 pF.
2. Board 9 matching:
    a. 0.5 pF – 50 cm
    b. 0 pF – 40 cm
    c. 0.2 pF – 45 cm (some readings were not received)
    d. 0.4 pF – 50 cm
    e. 0.6 pF – 55 cm
    f. 0.8 pF – 55 cm
    g. 1.0 pF – some intermittent behavior observed in the range of 60 – 70 cm
    h. 1.2 pF – some intermittent behavior observed in the range of 65 – 75 cm
    i. 1.5 pF – the board functions well at 70 cm; intermittent behavior still observed at 75 cm

3. Additional capacitance values may be tested with Board 9 to optimize its performance. However, this was not done because the working distance of Board 9 has been brought close to that of the ARS board used in the initial impedance matching process (70 cm).
4. Board 1 was used for the detailed antenna impedance matching tests that were previously performed. Eventually, LC components had been added and removed from the board so many times that the metal traces on the PCB began to peel off. After this occurred, the board only functioned correctly at extremely short distances (< 20 cm). The matching was then adjusted in an attempt to attain the original working distance again:
    a. 0.5 pF – 20 cm
    b. 0 pF – 25 cm
    c. 1.0 pF – 20 cm
    d. 0.2 pF – 25 cm
    e. 1.5 pF – < 20 cm

    From these results, it did not seem that tweaking the matching values would allow Board 1 to function anywhere near the distance that it originally did (70 cm). This board had seen a great deal of wear and tear, and perhaps some components had been damaged during repeated soldering. To minimize the time spent fixing the board, it was decided to simply fabricate a new one (extra printed circuit boards were available). The new board exhibited a different antenna impedance than the original; the best match yielded an operating distance of 65 cm with a 1.5 pF capacitor.
5. Board 10 matching:
    a. 0.5 pF – 45 cm

b. 0 pF – 35 cm

c. 1.0 pF – 55 cm

d. 1.5 pF – 65 cm

e. 2.0 pF – 65 cm

f. 1.8 pF – 65 cm

g. 1.3 pF – 65 cm

h. 1.6 pF – 65 cm

i. 3.0 pF – 55 cm

With readily available capacitor values, it appears that 65 cm is the best distance for Board 10. A 1.5 pF capacitor was chosen for the final matching because a good supply of them was available.

6. Board 3 matching:

   a. 0.5 pF – 55 cm

   b. 0 pF – 45 cm

   c. 1.0 pF – 60 cm

   d. 1.5 pF – 65 cm

   e. 2.0 pF – 65 cm

   f. 1.8 pF – 65 cm, very intermittent behavior observed at 70 cm

   g. 1.6 pF – 65 cm, some intermittent behavior observed in the range of 70 – 75 cm

7. Board 4 matching:

   a. 0.5 pF – 45 cm

   b. 0 pF – 30 cm

   c. 1.0 pF – 50 cm

d. 1.5 pF – 70 cm

8. Boards 7 and 8 were found to operate at 45 cm with matching values of 27 nH and 0.5 pF. This is the same behavior that was seen with Board 4. Considering that 1.5 pF proved to be a good match for Board 4, this value was tried with Boards 7 and 8 as well. In both cases, this resulted in a maximum working distance of 70 cm.
9. Board 6 did not work at all with the original matching values, which prompted a check of the circuitry. Upon examination, a bad connection was found on one pin of the SAW oscillator. After the problem was corrected, Board 6 was found to work at 65 cm with a 1.5 pF capacitor.

In summary, Table 5.3 contains the final antenna impedance matching values used on each board and their maximum operational distances when tested individually.

Table 5.3 Impedance Matching Results

	Antenna Impedance Matching		
Board	Inductance	Capacitance	Maximum Individual Working Distance
1	27 nH	1.5 pF	65 cm
2	27 nH	0.5 pF	75 cm
3	27 nH	1.6 pF	75 cm
4	27 nH	1.5 pF	70 cm
5	27 nH	0.5 pF	75 cm
6	27 nH	1.5 pF	65 cm
7	27 nH	1.5 pF	70 cm
8	27 nH	1.5 pF	70 cm
9	27 nH	1.5 pF	75 cm
10	27 nH	1.5 pF	65 cm

6.0 COMMUNICATIONS

The next step toward finishing the final prototype would be to evaluate the existing state of the non-interference protocol and update it if necessary to meet the project specifications – that is, to read ten boards in one tenth of a second or less. Previous work had already shown that several ARS boards work simultaneously with the initial non-interference protocol that had been designed for this prototype. However, no consideration had been given to the time required to perform a read cycle. The length of a read cycle is the amount of time required to power ten boards from a discharged state and obtain a temperature reading from each. First, the read cycle time using the existing software would be determined.

6.1 EXISTING PROTOCOL

The existing read cycle time was first examined using an oscilloscope. The supply voltage on an ARS board was monitored. The shape of the voltage waveform varies at different points in the read cycle (e.g., during power up, a sync pulse, etc.), and so the length of the read cycle can be determined by simply measuring the duration of the different variations. It was found that the total time for one read cycle using the existing software was approximately 1.8 seconds. By examining the transmitter PIC assembly code, it was determined that one read cycle is broken down as shown in Figure 6.1.

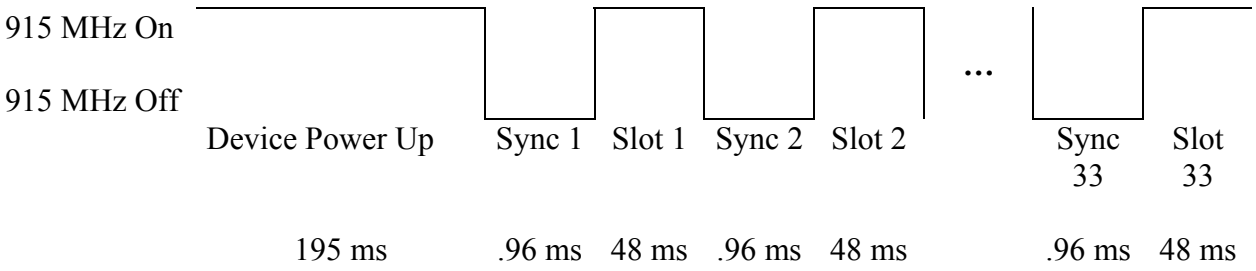


Figure 6.1 Initial Non-Interference Protocol Timing

The total time for one read cycle, therefore, is 195 ms + (48.96 ms x 33) ≈ 1.8 seconds. During each time slot, one ARS board transmits a single data byte. A 10-bit temperature reading requires two bytes of storage, so each ARS board must use two time slots to transmit its reading. With 33 time slots, the existing protocol allows a maximum of sixteen boards to be read. Note that the thirty-third time slot does not actually serve any purpose, and may be present due to an oversight when the software was written (e.g., a loop was supposed to iterate 32 times instead of 33).

Because the read cycle only needs to accommodate ten boards, 13 time slots can be ignored in the cycle length calculation. Therefore, a revised read cycle time using the existing protocol timing is 195 ms + (48.96 ms x 20) = 1.17 seconds. This still exceeds the required cycle length (100 ms) by more than ten times. The read cycle time of 100 ms was derived in the specifications by assuming that the protocol would allow ten boards to transmit 24 bits at 2,400 Baud with zero overhead (even time for framing bits as in RS-232 could not be afforded). Even though the data rate supported in the initial prototype three software is 9,600 Baud, the current protocol contains a lot of overhead built into it. For example, the time required to initially power the boards, send the sync pulses, and recover consumed energy after each byte is transmitted. This last source of overhead requires some additional explanation.

The existing software allocates two consecutive time slots for a given ARS board. When a byte is transmitted during a time slot, a significant amount of power is consumed on the ARS board (the transmitter component is the single greatest consumer of stored energy). In the limiting case where the ARS board is operating at its maximum distance from the energy source, the voltage level on the board following a byte transmission would be near the minimum that will allow the circuitry to continue operation. This means that sufficient time must be available to allow the board to recharge before the next transmission. Therefore, the time slot length in the initial protocol is approximately 48 ms, which is much longer than the time required to transmit one RS-232 encoded data byte at 9,600 Baud (~ 1 ms).

One proposed method for reducing the overall read time was to eliminate the large recovery time (~ 47 ms) between the slots. This could be achieved through "pipelining" the transmissions by having them occur in a round robin fashion, so that after one board sends a byte of data, it can recover while the other boards are sending their bytes. This could potentially reduce the overall time to 195 ms + (2 ms x 20) = 235 ms. Even if this method is applied, the device power up time must still be drastically reduced to meet the specified read cycle time.

As this partial solution was being considered, it was suggested that an attempt be made to make the boards function at a doubled data transmission rate (19,200 Baud). If this change would be possible, it would impact the read cycle time in an important way. Due to the bit time being halved, two bytes could be transmitted back-to-back (with a stop bit between) in approximately the same time as one byte could be transmitted at 9,600 Baud. So, whereas two time slots were needed for each board at 9,600 Baud, only one is needed at 19,200 Baud.

The number of time slots required for a single board to transmit its data illustrates a tradeoff between operational distance/power and time. By reducing the number of times slots

from two to one, ten boards may be read much more quickly. The large recovery delay between slots and the need for the "pipelining" approach described above are eliminated. Also, by halving the number of required time slots, the number of sync pulse delays is also halved which saves additional time. However, in general, the price paid is an increase in the power required for the transmission, and, hence, a decrease in the maximum operating distance. In this case there should not be a dramatic increase in required power because the Baud rate was doubled while the number of time slots was halved. Therefore, two bytes can be transmitted at roughly the same energy cost as previously needed to transmit one. Each board may transmit back-to-back, requiring a total time of approximately 195 ms + (.96 ms + (20 / 19,200) * 10) = 215 ms. Again, the 195 ms power up time still needs to be addressed.

6.2 DATA RATE IMPROVEMENT

At this point, significant algorithm changes were needed so that the ARS boards would function at 19,200 Baud. The RS-232 type signal sent to the 418 MHz transmitter is generated manually in software by toggling the voltage on a general purpose output pin. In the transmit portion of the existing assembly code, various delays were present to maintain a Baud rate of 9,600. Some delays took the form of defined constants that specified the number of iterations to spend in time-wasting loops between bit transmissions. However, simply modifying these constants would not suffice to double the Baud rate. More subtle delays were also present because there were various paths through the transmitter code. Different paths did not take the same time to execute because they contained differing numbers and types of instructions. This would need to be taken into account as well if the 19,200 Baud transmit routine was to be reliable in all situations.

Multiple paths were needed through the code because different delays were needed depending on the values of the current and previously transmitted bits. This would not be necessary if the 418 MHz transmitter behaved the same when turning on and off. However, during the development of the original software, it was found, by looking at the received 418 MHz RS-232 signal, that the transmitter takes a different amount of time to toggle from an off state to an on state than vice versa. This can be compensated for in software so that, at the 418 MHz receiver, all bit widths are consistent.

Some time was spent trying to determine how to modify the existing transmitter code for 19,200 Baud operation by taking these various sources of delay into account. After making a few changes to the software and examining their effects, it was decided that rewriting the transmitter code from scratch would likely prove quicker and more reliable than trying to modify the existing routine. This would eliminate any guesswork involved with changing the existing code and allow me to have a clear picture of the exact delay in the code. While the new software was being written and tested, the differences in transmitter switching times were observed using an oscilloscope. The new transmit routine also takes this problem into account and ensures that the transmitted bit times are consistent. The updated transmitter software is described in Section 7.2.

After modifying the software, several tests were performed where an ARS board was programmed to transmit a fixed bit pattern. Using an oscilloscope, the RS-232 signal output from the 418 MHz receiver was monitored, and the bit timing was evaluated. After several iterations of modifying the software and retesting, the resulting RS-232 signals at 19,200 Baud were satisfactory.

As an aside, one might wonder why an even higher Baud rate than 19,200 was not used. Written calculations have shown that the microcontroller, while running at a clock speed of 4 MHz, can support a much higher Baud rate than 19,200 in software. However, lab experiments with the 418 MHz receiver used for this project indicated that the maximum standard Baud rate supported by the receiver is 19,200. Still, this is more than sufficient to satisfy the data rate requirements for the current project.

The project specifications stated that future development phases for this device would require a data rate of at least 100K Baud. It can be shown that this data rate is theoretically possible using the current hardware. Consider the data rate of 100,000 Baud, which has a bit time of 10 μs. At this rate, the time to transmit one byte (not including the stop bit because it does not require the transmitter to be enabled) is 10 μs x 9 bits = 90 μs. The PIC12F675 operates at a 4 MHz clock frequency (.25 μs period), and hence 90 / .25 = 360 clock cycles would occur during the transmission of one byte at 100,000 Baud. There are plenty of cycles here to toggle the transmitter output for nine bits. However, the clock frequency of the microcontroller can vary with the supply voltage and temperature. For the range of supply voltages possible in this application, the internal oscillator frequency could lie anywhere between 3.8 MHz and 4.2 MHz [8]. The software to transmit at 100,000 Baud would be designed to work with a clock frequency of 4 MHz. The question now is whether or not this software would still work given these possible extremes.

The shortest possible clock period according to the datasheet is 1 / 4.2 MHz = .238 μs. The time allocated to transmit 9 bits is 360 clock cycles, following which the transmitter would be disabled for the stop bit. Assume that the value of each RS-232 bit is checked at the receiver in the middle of the bit time. Considering the beginning of the start bit to be time zero, the

middle of the ninth bit occurs at 10 µs x 8.5 bits = 85 µs. If it takes longer than 85 µs for 360 clock cycles to pass, then the 100,000 Baud software will work even with the shortest possible clock period. 0.238 µs x 360 = 85.68 µs; therefore, the software will still work.

A similar calculation shows that the software will work with the longest possible clock period, 1 / 3.8 MHz = .263 µs. With this clock cycle period, the ninth bit must start before 85 µs have passed; otherwise, the ninth bit will be lost. In software, the beginning of the ninth bit occurs after (360 / 9) x 8 = 320 clock cycles have passed. 0.263 µs x 320 = 84.16 µs; the ninth bit begins in time. These calculations have shown that there is much greater data rate potential in the current design than has been used for this project.

6.3 READ CYCLE TIMING

The data transmission Baud rate has been increased on the ARS boards. At this point, it is now possible to update the read cycle timing (Figure 6.1) to take this change into account and accomplish the goal of reading ten boards in a tenth of a second.

Due to the Baud rate increase, the number of time slots has been reduced to ten. This leaves three fundamental parts to a single read cycle as shown in Figure 6.2:

a. Power up time for the cycle

b. Ten sync pulse times

c. Ten transmit slot times

The task now is to determine the appropriate values for times *a*, *b* and *c*. The best value for each varies with a number of factors, particularly the distance of the sensor boards from the base station. This distance is a factor in how quickly the boards can harvest sufficient energy, which directly impacts the best values for times *a* and *c*. It is desired that the boards work at as

great a distance from the base station as possible, so values that are appropriate for that scenario will be sought.

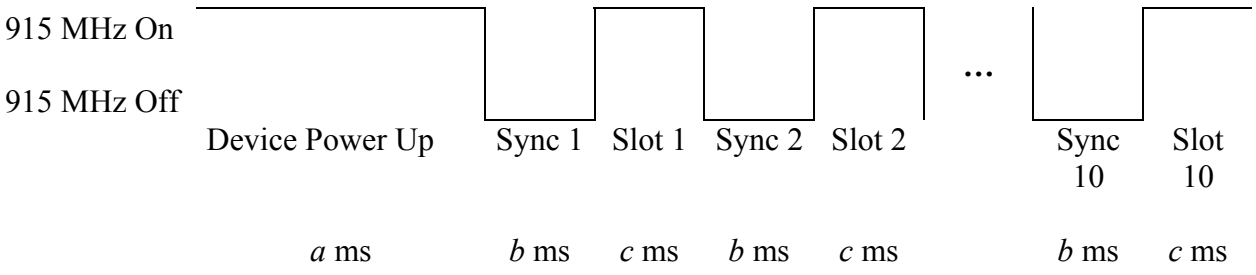


Figure 6.2 Generic Read Cycle Timing, Ten Time Slots

The sync pulses preceding each transmit slot are generated by briefly disabling the 915 MHz energy source. This prevents energy harvesting from occurring during the sync pulses. Therefore, the ARS device must consume previously stored energy to remain powered during these times. When the initial timing scheme was used, the energy consumption due to the sync pulses was negligible because each sync pulse of .96 ms was followed by a 47 ms transmit slot time (a long energy harvesting period). This is ample time to recoup the lost energy, even though the storage capacitor is in the slowest portion of its charging curve. In order to read ten boards in a tenth of a second, the transmit slot times had to be reduced to a minimum. The absolute minimum is the time required for the active board to execute its interrupt service routine and transmit 19 bits at 19,200 Baud. The transmission requires 19 bits * (1 / 19,200 bps) = .990 ms. By analyzing the ARS board assembly code and instruction execution times, a conservative estimate of the total time is 1.5 ms. From lab experimentation, it has been found that a 1.5 ms transmit slot time is not long enough to allow full recovery of the energy consumed during the

prior sync pulse. This is because the storage capacitor is almost fully charged and hence additional charge accumulates very slowly.

Due to the net loss of energy during each sync pulse/transmit slot pair, the voltage supply on the ARS board decreases across consecutive time slots (the voltage supply is proportional to the amount of stored energy). However, after several time slots occur, an equilibrium is reached whereby the amount of energy consumed during each sync pulse equals the amount recovered in the following transmit slot (the reason for this will be explained momentarily). The ARS board voltage supply also reaches equilibrium at this time due to its proportionality to the amount of stored energy. This equilibrium voltage is less than the voltage reached on the ARS board when a 47 ms transmit slot time is used. However, laboratory testing has shown that the equilibrium voltage achieved by using a 1.5 ms transmit slot time is greater than 95% of the voltage available when using a 47 ms transmit slot time (i.e., the voltage supply reduction caused by using the shortened transmit slot time is minimal). This percentage will vary with how quickly a board can harvest energy, which is a function of several variables, including its distance from the base station and the optimality of its antenna impedance matching.

The oscilloscope capture of Figure 6.3 illustrates the equilibrium phenomenon. The screen capture shows the supply voltage on an ARS board versus time. The time slots are indicated on the figure. The voltage is constant at the beginning of the waveform (label A on the figure) because the device is still in the power up phase and the storage capacitor is fully charged. The first drop in voltage (label B) corresponds to the energy loss due to the first sync pulse, and the subsequent small increase in voltage (label C) is due to the energy harvesting that occurs during the following transmit slot. A net voltage loss occurs during the first time slot, i.e.,

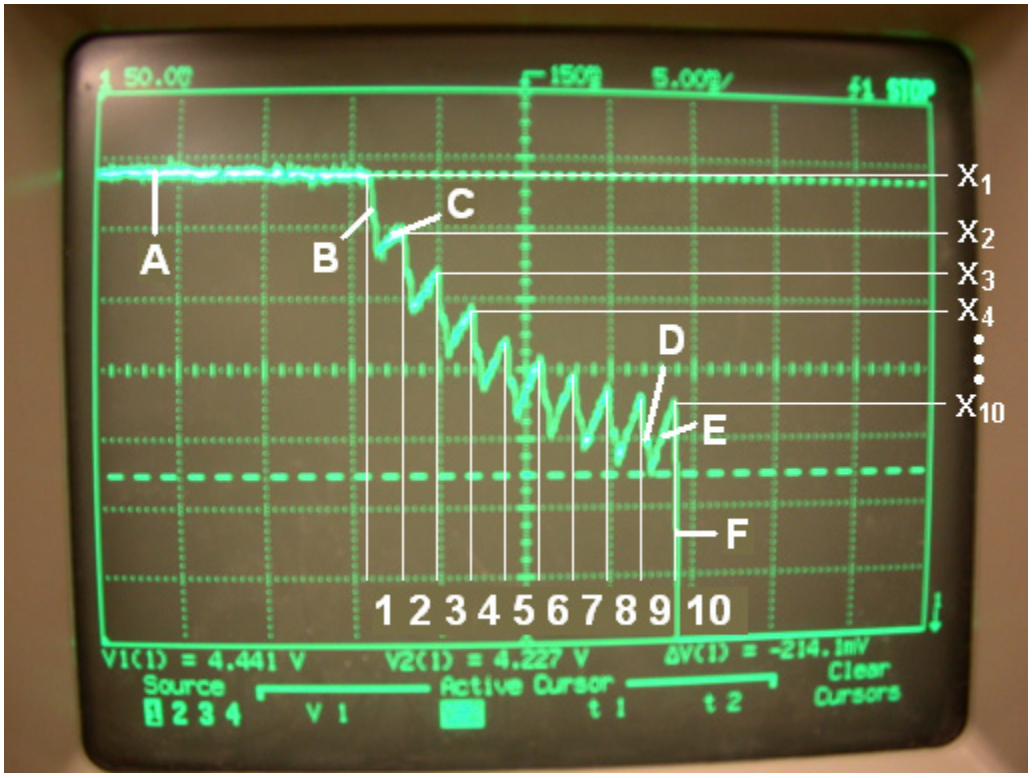


Figure 6.3 Voltage Consumption During Sync Pulses

A – C. During the subsequent time slots, the drops in voltage during the sync pulses are approximately equal, but more voltage is recovered with each transmit slot (e.g., x_1-x_2 > x_2-x_3 > x_3-x_4 > … > x_8-x_9 > x_9-x_{10}). This increase in the amount of recovered voltage is due to the storage capacitor being recharged more quickly than during earlier transmit slots. By the ninth sync pulse (label D), the amount of voltage recovered during the following time slot (label E) is very close to equaling the amount lost during the sync pulse. The total voltage loss due to the nine sync pulses is only 0.164 V / 4.441 V = 3.7% of the power up voltage (calculated by comparing the voltage before the first sync pulse to that before the tenth). Furthermore, very little voltage will be permanently lost due to additional time slots because equilibrium has been reached. These results show that greatly reducing the time slot length (from 48 ms to 1.5 ms, or

by 96.9%) has a relatively small impact on the available supply voltage (reduced by 3.7%). This loss is outweighed by the benefit of greatly reducing the read cycle time as required in the project specifications. The results validate the decision to reduce the time slot length from an energy harvesting perspective by confirming that doing so will not adversely impact the ARS board supply voltage. As a final note on Figure 6.3, this particular ARS board transmitted a temperature reading during the tenth time slot. This caused the rapid drop in voltage at the far right of the capture (label F).

The question is: how do these observations affect the choice of read cycle times *a*, *b* and *c* from Figure 6.2? In summary, they indicate that it is desirable to use the smallest working sync pulse width. A shorter sync pulse width not only saves overall read cycle time, but it leads to less net energy loss during each sync pulse/transmit slot pair. Also, having a longer transmit slot time is clearly preferable as this allows more time for energy harvesting between each sync pulse. These results assist in dictating how the read cycle timing should be determined for the application of reading ten boards in 0.1 seconds.

- First, the sync pulse time *b* should be minimized.
- Second, the power up time *a* must simply be long enough to fully charge each board. This time will vary based on the distance of each board from the base station but can be experimentally determined for a given situation. ARS board charging time data gathered in the lab is presented in Table 6.1, which will be discussed shortly.
- Finally, whatever time is left over in the tenth of a second can be divided among the ten transmit slots. Remember that each slot must be at least ~1.5 ms long for proper operation. Therefore, if this constraint cannot be met for a given base station distance and *a* cannot be

reduced to a value that still allows each board to obtain sufficient charge to operate, then the boards cannot be read in 0.1 seconds at that distance.

Consider the sync pulse delay time. The ARS board energy harvesting and sync pulse circuitry is shown in Figure 6.4. The sync pulse must discharge the 15 pF capacitor C1 on the ARS board. The voltage on this capacitor is connected to an I/O pin on the microcontroller, and the state change of this digital input triggers an interrupt that awakens the processor from the sleep state. Diode D3 isolates the main energy storage capacitor C2 from this process, allowing capacitor C1 to discharge without affecting the charge previously stored on C2. Capacitor C1 is connected to ground through a 1 MΩ resistor (R5), giving an RC time constant of 15 μs for the discharging of capacitor C1. The capacitor is sufficiently discharged after 3RC = 45 μs. However, lab experiments have shown that using this sync pulse delay in the base station software is not enough to trigger the ARS board. Therefore, additional factors must be playing a role; for example, a possible delay in switching the base station transmitter off. It appears from lab experimentation that a sync pulse delay in the base station software of approximately 455 μs is the minimum acceptable value. For added reliability, it might be better to use a slightly higher value if possible given the time constraints of the application.

Next, the time required to charge an ARS board was examined. An oscilloscope probe was attached to the voltage supply on an ARS board to observe the time required for the voltage to increase to 95% of its maximum value at various distances from the base station. The results are shown in Table 6.1. The minimum operational voltage for the ARS board is somewhere around 3 V or slightly less. Therefore, considering the charging time data in Table 6.1, a device power up time around 75 – 80 ms would be appropriate in the read cycle timing. When

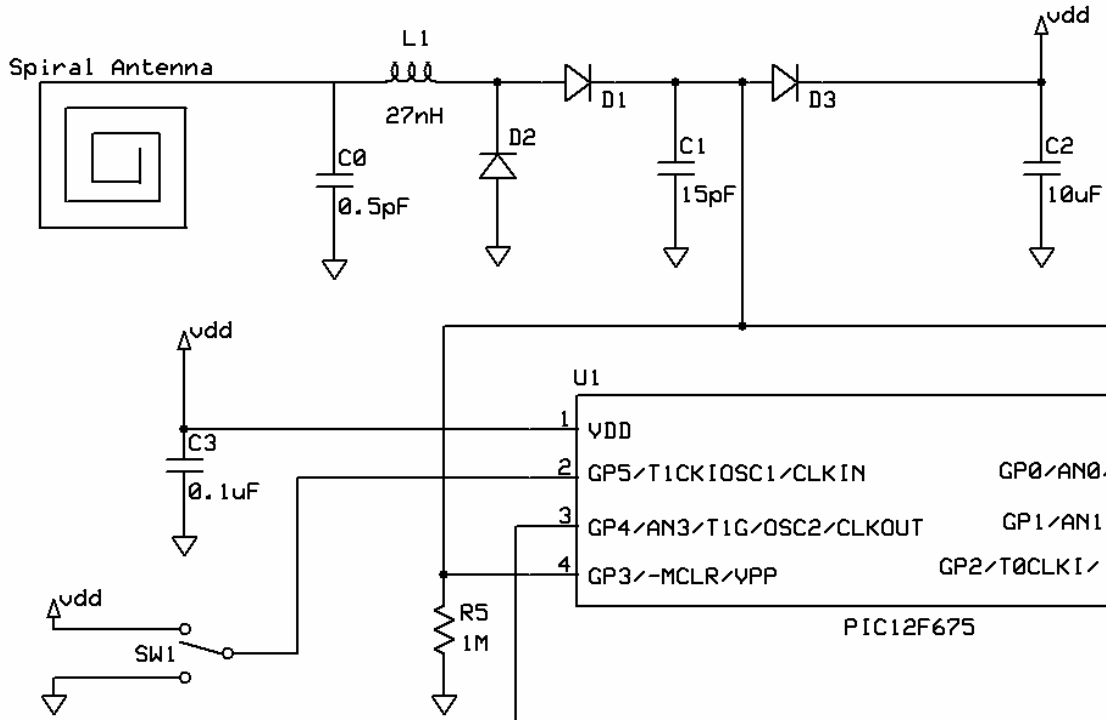


Figure 6.4 ARS Board Energy Harvesting and Sync Pulse Circuitry

Table 6.1 ARS Board Charging Times at Various Distances

Distance From Base Station (cm)	Maximum Voltage (V)	Approximate Charging Time To 95% Of The Maximum Voltage (ms)
80	4.75	39
75	4.69	42
70	4.50	43
65	2.53	77
60	1.69	90
57	2.81	75
56	3.06	68
55	3.38	63
50	4.31	50
40	4.75	44

examining the data in Table 6.1, it is interesting to note that the longest charging times occur in the middle of the distance range. Multiple charging time measurements have been obtained that confirm this behavior. One possible explanation for these results is that certain behaviors are a function of the ¼, ½ and ¾ RF wavelengths in the near field, and the largest charging times occur at approximately 1¾ wavelengths from the base station antenna [9].

A final note on the read cycle timing concerns the amount of time that the 915 MHz energy source is disabled between consecutive cycles. This time is not considered to be a part of the 100 ms read cycle and therefore can be chosen arbitrarily. However, the time between read cycles must be long enough to allow the ARS board energy storage capacitor to discharge sufficiently (enough so that the microcontroller resets due to low supply voltage). An ARS board cannot recognize a new read cycle until its software is reset. A delay of 390 ms between read cycles was found to be acceptable when testing the ARS boards in the laboratory.

A problem arose during testing in the lab whereby the boards seemed to require some minimum distance from the base station in order to function properly. Obviously, the supply voltage on an ARS board increases as it is moved closer to the base station. Therefore, it seemed that this problem was due to the supply voltage becoming too high for the microcontroller, and a method to limit the voltage was investigated. It was later determined that this problem was actually caused by an insufficient length of the delay between read cycles. As an ARS board is moved closer to the base station, the amount of time necessary to discharge the energy storage capacitor increases along with the supply voltage. The 390 ms delay between consecutive read cycles, although sufficient at greater distances from the base station, was not long enough to allow the microcontroller to reset when the board was moved closer. Therefore, the ARS boards appeared to stop working at shorter distances, although they simply were not resetting for new

read cycles. Through a trial and error process, a longer delay between read cycles was found that corrected this problem. When the delay between read cycles was increased to 1.95 seconds, the boards were found to have no limitation on their minimum operational range from the base station.

7.0 SOFTWARE

The research concerning updating the non-interference protocol is complete, and now the software running on the individual ARS boards as well as the base station must be designed to implement the new read cycle timing. This section details the modifications that were made to the existing prototype three software.

7.1 GENERAL ARS BOARD SOFTWARE

Changes to the ARS board initialization routine needed to be made due to the hardware modification that had been performed to improve the temperature reading accuracy (see Section 5.1). This hardware update involved modifying the pinouts of the microcontroller as well as enabling the A/D converter external voltage reference feature.

The values of some configuration registers needed to be updated to reflect the pinout change. The TRISIO (GPIO tri-state) register contains a bit for each general purpose I/O pin designating whether the pin is an input or output. The 418 MHz transmitter enable output was moved from GPIO2 to GPIO4 because the A/D converter reference input must be on GPIO1 and shorter PCB traces are needed if the thermistor circuit voltage supply is output from GPIO2 (this is because the GPIO1 and GPIO2 pins, which must be tied together, are physically close to one another as shown in Figure 7.1). The TRISIO register was updated to make GPIO4 an output and GPIO1 an input. The ADCON0 (A/D control) register is used to configure several aspects

of the internal A/D converter. The VCFG bit in this register was changed to indicate that the V_{REF} pin (GPIO1) should be used as the voltage reference instead of V_{DD}. Finally, the ANSEL (analog select) register contains bits that are used to select which of four GPIO pins will be used as analog inputs to the A/D converter (four channels are available on the PIC12F675 although only one is used here). The bit corresponding to GPIO1 was changed to an analog input because it is now receiving the analog reference voltage.

The pseudocode in Figure 7.2 describes the overall behavior of the ARS board software.

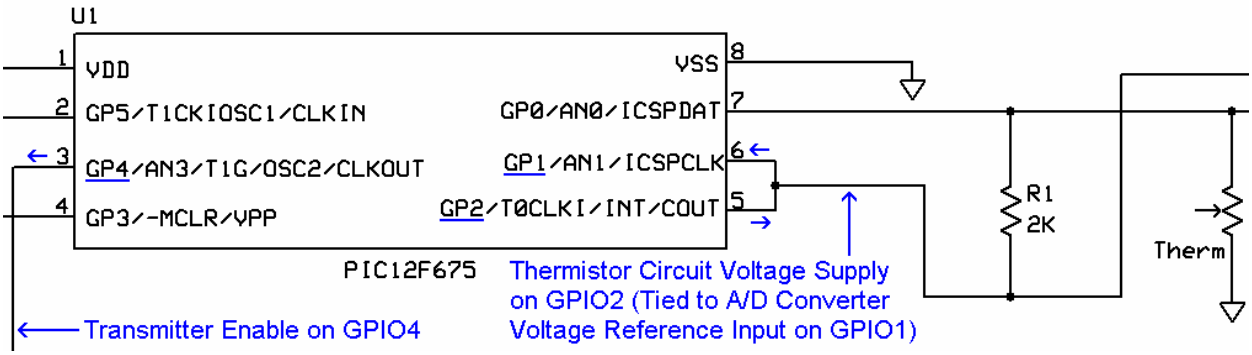


Figure 7.1 Updated PIC GPIO Pinouts

7.2 TRANSMIT ROUTINE

As mentioned in Section 6.2, the transmit routine was redesigned to increase the data channel Baud rate from 9,600 to 19,200 bits per second. A flowchart description of the updated code is shown in Figure 7.3.

Different paths through the flowchart in Figure 7.3 are taken based upon the values of the previous, current and next bit to be transmitted. This is necessary because lab

```
// Defined constants
DEVICE_ID   0       // Unique sensor identification number
                    // Can be 0 – 15 in current implementation but is scalable

// The init code runs once during the device power up period
Init:
        Initialize microcontroller configuration registers (sets up general purpose I/O pins, A/D
        converter and other microcontroller-specific features)
        Enable sync pulse interrupt
        // Clear time slot counter
        timeSlot = 0
Main:
        Enter sleep mode
        // Microcontroller will awaken when sync pulse interrupt occurs
        goto Main

// Sync pulse interrupt service routine
SyncISR:
        Save current processor state
        If (Interrupt source is not sync pulse) {
                // No other interrupts should be enabled
                goto EndISR
        }
        If (DEVICE_ID == timeslot) {
                // This is the correct time slot to transmit the temperature reading
                Access A/D converter and acquire 10-bit reading
                Form low and high bytes using A/D reading and DEVICE_ID
                Transmit low byte
                Transmit high byte
        } Else {
                // This time slot belongs to another sensor
        }
        // Increment time slot count
        timeSlot++
EndISR:
        Restore previous processor state
        Return from interrupt
        // “Return from interrupt” automatically re-enables sync pulse interrupt
        // Processor will return to sleep mode until the next interrupt occurs
```

Figure 7.2 ARS Board Pseudocode

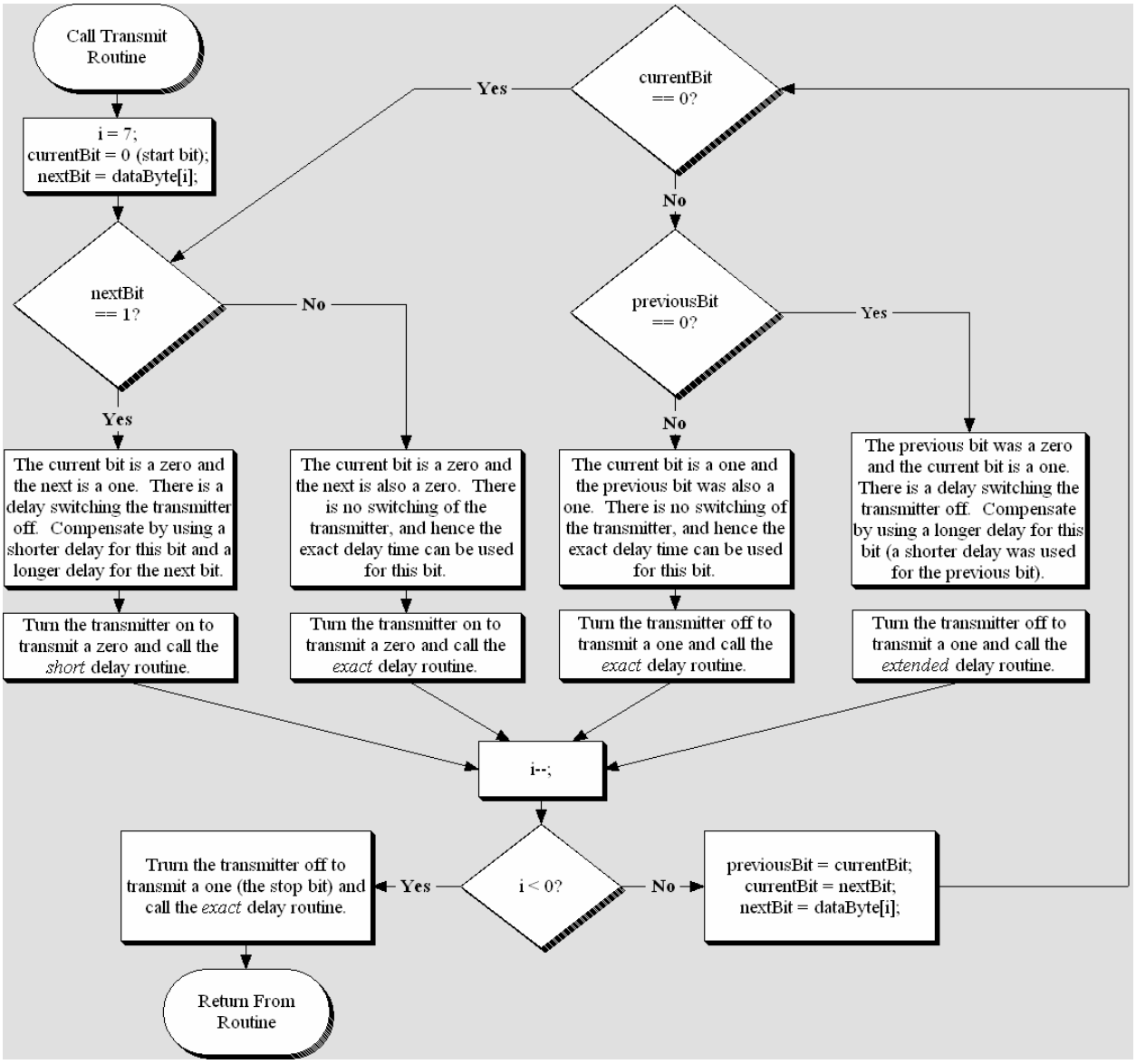


Figure 7.3 Transmit Routine Flowchart

experimentation indicated that the transmitter requires a significant amount of time (approximately 10 μs) to turn off after the proper signal is output from the microcontroller. However, a delay was not observed when the transmitter was turned on. This means that a software remedy is required to provide consistent bit widths in all cases.

A delay only occurs when the transmitter is switched to the disabled state. The transmitter is enabled when logic zeros are transmitted and disabled for logic ones. The new transmit routine detects when the current bit to be transmitted is a zero and the next bit is a one. At the end of the current bit time, the transmitter will need to be disabled. Due to the transmitter turn-off delay, the current bit time as output from the microcontroller to the 418 MHz transmitter should be ended short. The turn-off delay will provide the remainder of the bit time as seen at the 418 MHz receiver. A short bit time delay is used in the software to handle this case.

Because this bit time has been ended short from the perspective of the microcontroller, the next bit time must be lengthened by the same amount to prevent future bit transitions from occurring too early (in other words, to maintain the synchronization provided by the start bit). The software provides an extended bit time delay in this case when the current bit to be transmitted is a one and the previous bit was a zero. For all other situations, an exact bit time delay of 52 μs is used.

The assembly language software was written taking the instruction execution times of each path through the flowchart into account so that the paths consistently require ~52 μs for each bit time (except for the intentionally differing short and extended bit times). Although some bit time error is acceptable with the RS-232 type interface at the base station receiver, the effort has been made to maintain the correct bit times in all cases. To test the correctness of the code while taking the transmitter behavior into account, an oscilloscope was connected to a 418 MHz receiver to view the received RS-232 waveforms for several test transmissions. Additional tweaking of the software delays was performed based on this feedback. The final result was consistently valid 19,200 Baud RS-232 waveforms at the receiver.

7.3 BASE STATION SOFTWARE

The base station software controls the 915 MHz energy source that is used to power and synchronize the ARS boards. The microcontroller running in the base station is a PIC16F870. The 915 MHz transmitter is enabled or disabled by setting the appropriate digital output on one of the microcontroller I/O pins. The physical construction of the base station with the PIC embedded inside was performed prior to my involvement with prototype three. Existing software for the base station was available, but it supported the initial read cycle timing scheme (Figure 6.1). The delay values in the software needed to be modified to use the updated read cycle timing and allow ten boards to be read in a tenth of a second. A simple pseudocode description of the base station software is provided in Figure 7.4.

The various delays referred to in the pseudocode are implemented in assembly language using loops. The counter value for each loop determines the length of the delay. The body of each loop requires three instruction cycles per iteration [10]. The clock frequency of the microcontroller is 4 MHz, and four clock cycles comprise a single instruction cycle. Therefore, each instruction cycle is 1 μs in length, and each iteration of a delay loop consumes 3 μs. A comparison of the original and updated read cycle timing delays is provided in Table 7.1.

The updated time slot delay value of 1.5 ms was derived previously by taking into consideration the amount of time needed to transmit 19 bits at 19,200 Baud and the software overhead of the ARS board transmit routine. Prior lab experimentation had shown that a sync pulse delay of 455 μs was the minimum sufficient for triggering the ARS boards. However, while testing the updated read cycle timing with the ten ARS boards, it was observed that the boards returned valid temperature readings more consistently when a longer sync pulse delay was used. After testing the ARS boards with a number of sync pulse delay values, the time of

1.2 ms was decided upon because it worked well in the lab and it was not so large as to greatly reduce the time available for the power up delay. Ten sync pulse and time slot delays are needed, and they consume a total of 27 ms of the read cycle. Therefore, the remaining 73 ms is dedicated to device power up.

```
Init:
        Initialize microcontroller configuration registers (set up I/O pins, disable interrupts, etc.)
Start:
        // This is not part of a read cycle, but actually takes placed between cycles
        Disable transmitter
        Perform "Brown-Out" Delay          // A very long delay (~ 400 ms) to allow the ARS
                                           // boards to discharge and reset between read cycles
        // This is the beginning of a new read cycle
        // Device power up phase
        Enable transmitter
        Perform "Power Up" Delay

        // Sync pulses and time slots
        // numSlots = 33 in the original code, changed to 10 for the final version
        for(i = 0; i < numSlots; i++) {
                Disable transmitter
                Perform "Sync Pulse" Delay
                Enable transmitter
                Perform "Time Slot" Delay
        }

        goto Start
```

Figure 7.4 Base Station Pseudocode

Table 7.1 Base Station Software Delays

	Original Code		Updated Code	
	Number of delay loop iterations	Length of delay (ms)	Number of delay loop iterations	Length of delay (ms)
Power Up Delay	65,025	195.075	25,000	75.000
Sync Pulse Delay	320	0.960	400	1.200
Time Slot Delay	16,000	48.000	500	1.500

The length of the power up delay in software is 25,000 loop iterations, which would be 75 ms. However, only 73 ms are available in the read cycle. While using this delay value, an oscilloscope probe was attached to the voltage supply on an ARS board to obtain a picture of the charging curve. The oscilloscope capture showed that the board was only charging for approximately 71 ms. One possible explanation for this is that the 915 MHz transmitter might require some additional time to start from a completely powered-down state as is the case between read cycles. This is not the case for the sync pulses which require very brief disengagements of the transmitter and therefore do not exhibit this delay. The read cycle only begins when the transmitter turns on and the boards begin charging, so the "missing" part of the power up delay is not counted against the read cycle time. Therefore, the power up phase requires less than 73 ms, and hence fits into the 100 ms read cycle. The final non-interference protocol timing is shown in Figure 7.5.

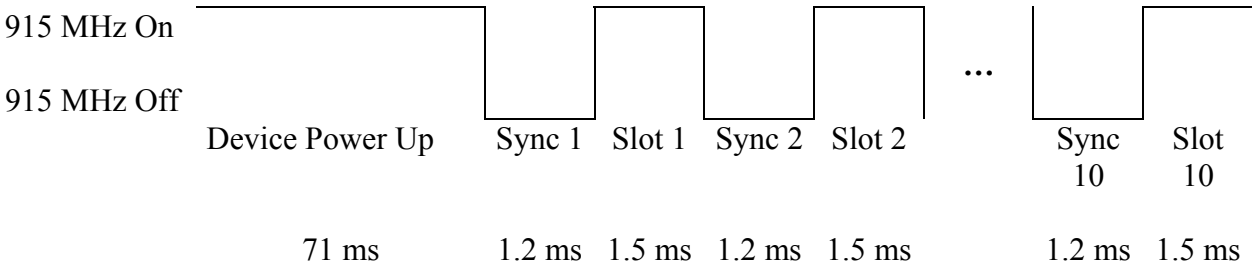


Figure 7.5 Final Non-Interference Protocol Timing

8.0 RESULTS

Ten ARS boards were completely fabricated, and all software was updated to use the revised non-interference protocol. What remained at that point was to verify that all ten ARS boards return correct temperature readings in 100 ms. Before presenting the test results, an overview of the demonstration setup is provided.

8.1 DEMONSTRATION SETUP

The entire test system consists of the base station, ARS boards, 418 MHz receiver and a PC. The functionalities of the base station and ARS boards have already been presented. Figure 8.1 shows the testing setup where the ARS boards are positioned in front of the base station patch antenna. What remains to be discussed is the formatting of the 418 MHz temperature reading transmissions and how this information is presented to the user.

A simple framing mechanism was employed for the transmissions between the sensor boards and the 418 MHz receiver. When a sensor's time slot is reached, it uses the 10-bit A/D converter internal to the PIC12F675 microcontroller to sample the voltage across a thermistor. These ten bits of data (TEMP[9...0]), as well as the sensor's unique identification number, are packed into a two-byte transmission as shown in Figure 8.2.

The sensor identification number is currently four bits long to accommodate ten devices. However, the number of bits for the ID can be increased to allow hundreds of sensors to be used

with the non-interference protocol. Alternatively, if the receiver were to be aware of the number of the current time slot, then the sensor identification number would not need to be transmitted at all. The most significant bit is used to identify which byte (low or high) has been received. This simple mechanism allows the receiver to resynchronize itself with the incoming bytes in the case of a missing or damaged transmission.

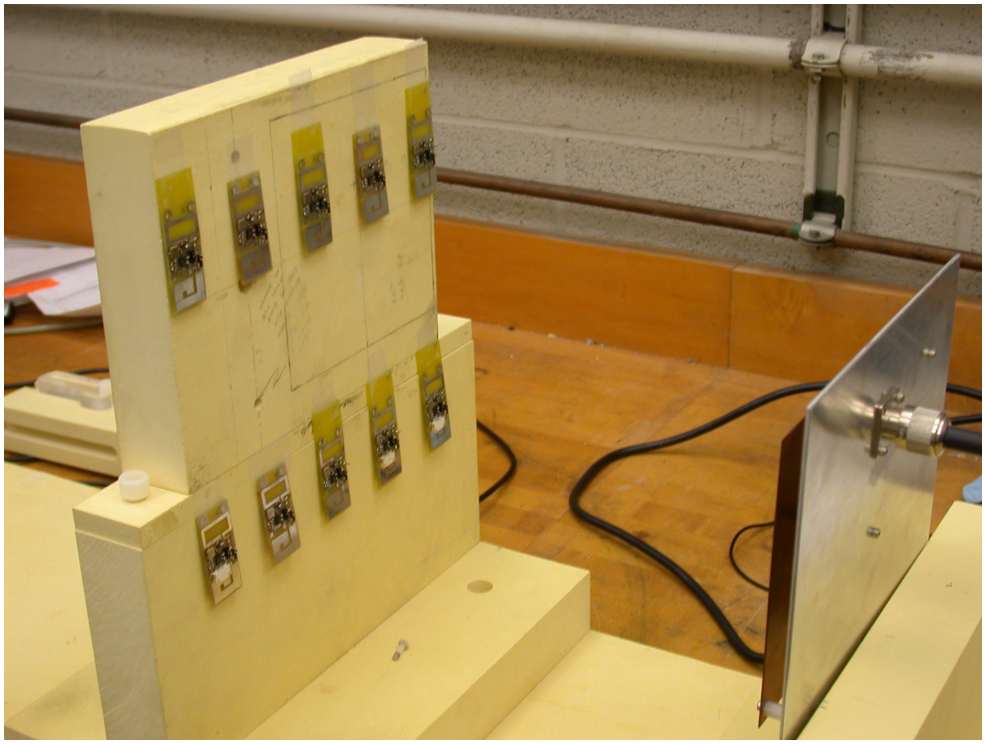


Figure 8.1 ARS Boards and Base Station Antenna

First (Low) Byte

Value	0	Sensor Identification Number				TEMP[2…0]		
Bit	7	6	5	4	3	2	1	0

Second (High) Byte

Value	1	TEMP[9…3]						
Bit	7	6	5	4	3	2	1	0

Figure 8.2 Temperature Data Framing

The two bytes are transmitted using amplitude shift keying (ASK) at a frequency of 418 MHz. Again, the transmission protocol is simply RS-232 (19,200 Baud, eight data, no parity, one stop bit). The 418 MHz receiver simply changes the medium of the RS-232 transmission from RF to a standard DB9 serial cable that is run to a PC. It does not perform any intelligent processing of the data. On the PC, the temperature readings may be parsed and then processed in any way necessary. To save computation, and hence power, on the remote device, the conversion between raw A/D converter reading and actual temperature (degrees F, C, etc.) occurs on the PC side. The pseudocode in Figure 8.3 describes the behavior of a simple PC receiver program. For the tests performed in the lab, a short MATLAB receiver program written previously was used to capture and display the raw A/D converter readings.

```
        Open serial port
        lowByteReceived = false
GetNextByte:
        Wait for byte on serial port
        Get byte from serial port
        If (MSB is “0”) {
        // Low byte received
        Parse byte, store ID number and lowest 3 bits of temperature
        lowByteReceived = true
        } Else {
                // High byte received
                If (lowByteReceived == true) {
                        Parse byte to obtain highest 7 bits of temperature
                        Process temperature reading as necessary and display to user
                        lowByteReceived = false
                } Else {
                        // No accompanying low byte was received; error
                        // Disregard the byte
                }
        }
        Goto GetNextByte
```

Figure 8.3 PC Receiver Program Pseudocode

8.2 INTERPRETATION OF A/D CONVERTER READINGS

The thermistor used on the ARS boards (Panasonic part number ERT-J1VT202J) has a resistance of 2 kΩ at 25° C (77° F) and is used in a simple voltage divider circuit with a resistor of the same value [11]. The A/D converter reference voltage is the supply to the divider circuit. The resistance of the thermistor may be determined by measuring the voltage drop across it, and this resistance directly corresponds to a temperature reading according to the thermistor specifications.

Consider that V_{ref} is the voltage supply to the divider circuit and V_{th} is the voltage drop across the thermistor. Let the resistance of the thermistor be R_{th}. According to the voltage divider equation, the voltage drop across the thermistor is $V_{th} = V_{ref}[R_{th} / (R_{th} + 2{,}000)]$. From the perspective of the 10-bit A/D converter which directly measures the thermistor voltage, $V_{th} = V_{ref}[AD_{reading} / (2^{10} - 1)]$, where $AD_{reading}$ is the 10-bit output value ranging from 0 to $(2^{10} - 1)$. Equating these expressions for V_{th}, we have:

$$V_{ref}[R_{th} / (R_{th} + 2{,}000)] = V_{ref}[AD_{reading} / (2^{10} - 1)]$$

Canceling V_{ref} on both sides and rearranging, this can be solved for R_{th}:

$$R_{th} = (2{,}000 * AD_{reading}) / [(2^{10} - 1) - AD_{reading}]$$

According to this equation, the resistance of the thermistor can be determined solely from the value of $AD_{reading}$.

The resistance values for many temperatures are provided in [11]. Using the above equation, the resistance values may be changed to A/D converter readings, thereby obtaining several data points relating A/D readings and temperature. The graph in Figure 8.4 is obtained by taking these data points and plotting them in Microsoft Excel.

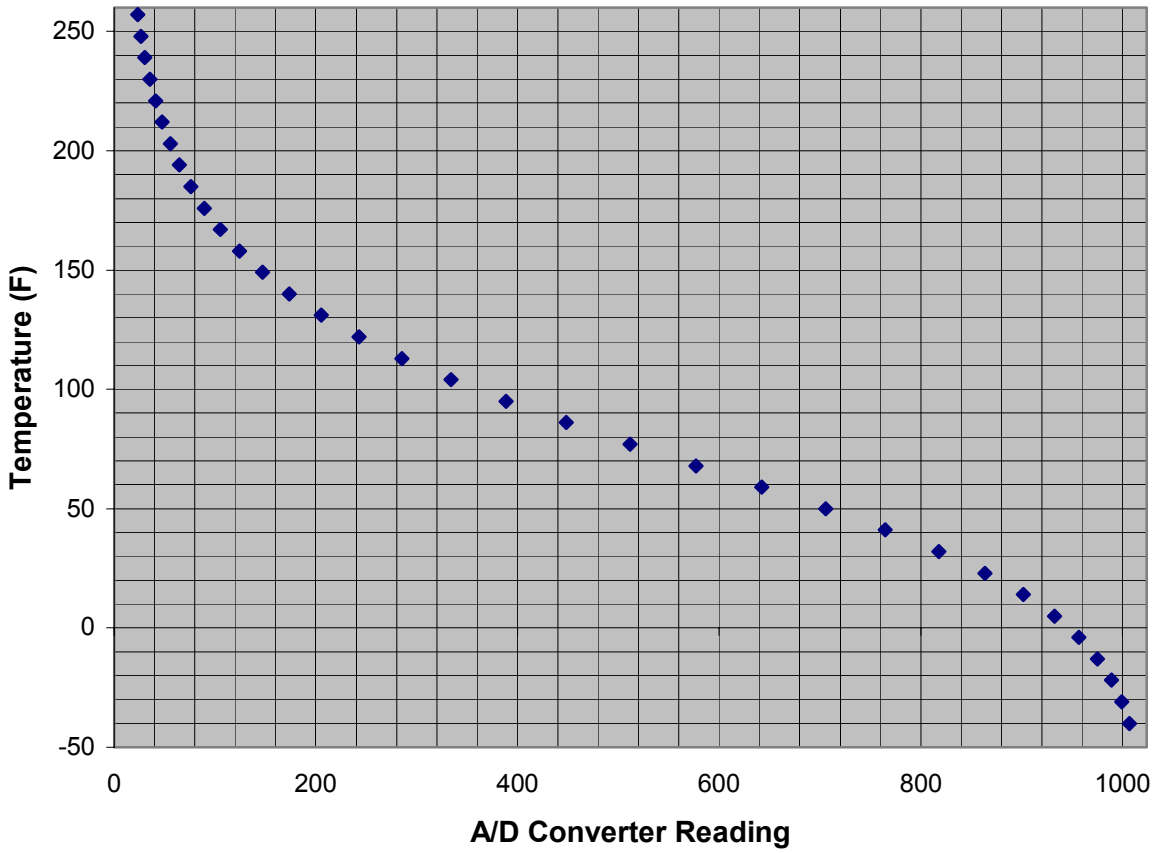


Figure 8.4 Relationship Between A/D Converter Readings and Temperature

8.3 PROTOTYPE THREE DEMONSTRATION

Using the lab setup described above, the ten ARS boards were read simultaneously at a distance of 40 cm from the base station. This is a significant reduction in operational distance

from the single board case, but this is to be expected because multiple devices must be within the limited field pattern at a point with sufficient distributed energy to activate all boards simultaneously. The MATLAB screen capture in Figure 8.5 shows successive read cycles in which all ten boards are correctly read. Each line of output corresponds to a received 16-bit temperature transmission. The device ID for the reading is displayed along with the raw A/D converter value. All of the A/D readings are in the range of 520 – 530. Referring to Figure 8.4, it is seen that this range corresponds to temperature readings between 75.9° F and 74.5° F. These values are correct given the lab thermostat reading at the time of the test and the individual heating of each board due to the RF energy field at its particular location.

It is also necessary to obtain proof that the read cycle is indeed occurring in a tenth of a second. To do this, an oscilloscope probe was attached to the voltage supply on one of the ARS boards during a read, which yielded the waveform in Figure 8.6. The left part of the waveform, from the left cursor until just before the horizontal center of the screen, is the power up portion of the read cycle. The large energy storage capacitor on the sensor board is charging during this time. The drop in voltage at the horizontal center of the screen occurs when this board (identification number 2) transmits its temperature reading. The slow increase in voltage following this drop is due to the additional energy harvested during subsequent time slots. Finally, the steady drop in voltage at the right is the power down period following the read cycle. The cursors show that the total time from the power up to power down phases is just less than 100 ms, which fulfills the read cycle timing requirement.

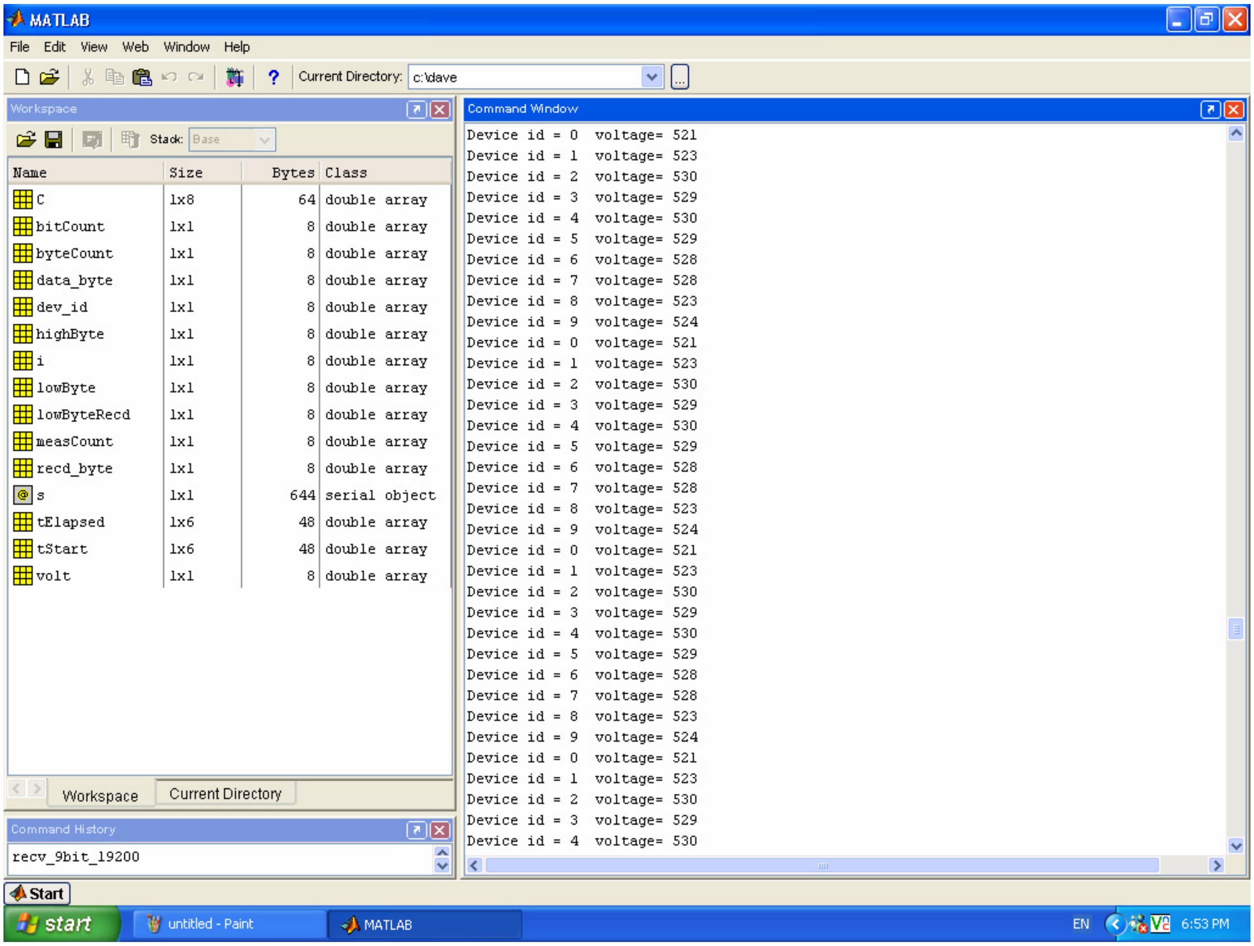


Figure 8.5 Demonstration MATLAB Screen Capture

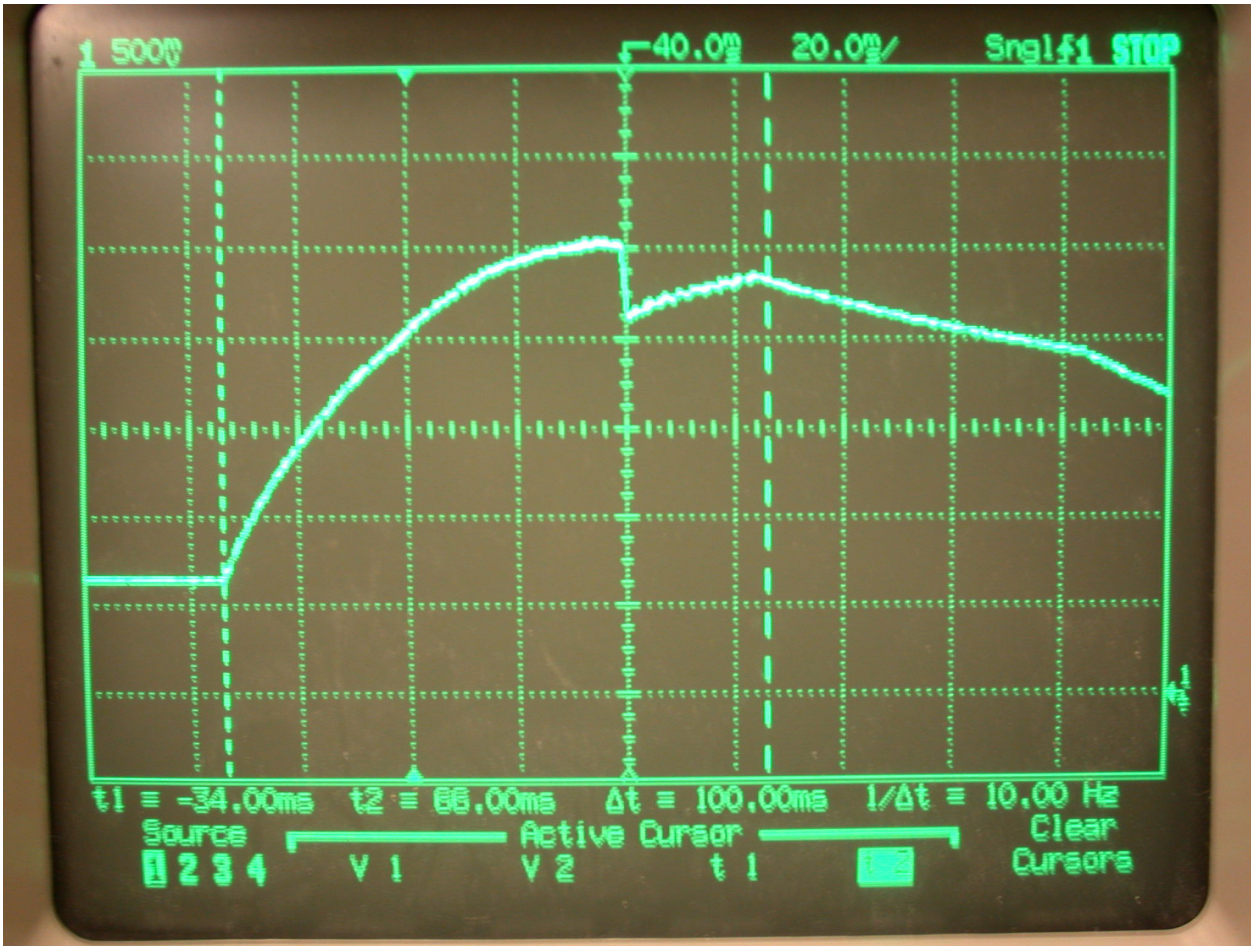


Figure 8.6 Oscilloscope Capture for Read Cycle Timing Verification

9.0 CONCLUSIONS

This project may be considered a success if all of the contractual specifications have been met. Looking back over the specific operational requirements of the device as stated in the problem statement, it is seen that all of the criteria have been met. The final prototype sensor board has a form factor of 0.864" by 1.654", which is smaller than the required size. A complete embedded non-interference protocol was indeed developed, and the necessary software was designed and implemented in both the sensor boards and the base station. This protocol is easy to understand and very easily extendable to accommodate additional devices simply by adding additional time slots. It was chosen over the protocol developed in prototype two because the sync pulse mechanism continually synchronizes the ARS boards so that all devices know exactly when each time slot begins. In the prototype two method, each device determines the start of its time slot based on a software delay performed after powering up. If all sensor boards are not powered simultaneously, or if there are slight differences in the microcontroller clock frequencies, then the devices may lose their synchronization, especially those transmitting during later time slots. Even though the final protocol has more overhead compared to the simpler one developed for prototype two, the benefit of its enhanced synchronization method makes it much more suitable for supporting hundreds of devices as required.

Ten sensor boards were successfully fabricated and tested simultaneously in the lab using the non-interference protocol. Valid temperature readings were returned from all boards within a tenth of a second, and no interference problems were observed. Also, the Baud rate supported by

the sensor boards for data channel transmissions (19,200 Baud) exceeds the specifications (2,400 Baud).

In addition to the project specifications being met in full, a second project goal has been attained as well. Through this research, my research and engineering skills have been strengthened. In retrospect, I can say that this experience has been a very beneficial one for me, during which I have learned new skills and strengthened old ones. The opportunity to work on a project with real-world implications and in an essentially independent manner was a valuable one. From a hardware perspective, I became familiar with a number of skills with which I had no prior experience, including printed circuit board layout and surface mount component hand and reflow soldering. Also, I had never worked with RF devices in a lab environment, and I gained experience in the subtleties of testing these devices. Considering the software side of the project, I greatly improved my knowledge of Microchip PICs and their configuration. Also, I reviewed and strengthened my skills in C, assembly language and MATLAB programming. Finally, I consider the general area of embedded system research and development to be of particular interest to me, and this project fits well into that category. An ARS board, consisting of a microcontroller and peripheral circuitry, is a simple embedded system that involves hardware and software co-design. I very much enjoyed working on this project and bringing it to a successful conclusion.

9.1 FUTURE CONSIDERATIONS

A number of issues remain to be examined during future development of the NASA temperature sensor. An investigation into the operational environment of the final product was

beyond the scope of the first development phase addressed by this research. However, as research on the device continues, it will certainly be important to determine what unique challenges might be presented to its operation when it is deployed on a spacecraft. For example, it is possible that the sensor device might be subjected to very high temperatures, or that the materials used in the construction of the spacecraft panels could greatly attenuate the RF signals transmitted to and from the temperature sensors. These operational challenges may be better understood and addressed once the details of the final environment are known (e.g., where the sensors are placed on the spacecraft panels, the relative positioning of the RF energy source and data receiver, etc.).

Also, future research could verify that the supply voltage on an ARS device does indeed reach equilibrium after several time slots have occurred. This conjecture has been made based upon the oscilloscope capture in Figure 6.3. It appears from the figure that the amount of voltage lost during the ninth sync pulse is nearly identical to the amount recovered during the following transmit slot. To verify that equilibrium is truly reached, the supply voltage on the ARS device can be monitored during the passage of many additional time slots.

While the operational distance of the sensor device was not specified for the first development phase, additional research could focus on improving the maximum operational distance of the sensors from the base station. One approach would be to further reduce the power consumption of the device. For example, the output power of the 418 MHz transmitter could be reduced and a more sensitive receiver used to detect the weaker signal. As the trend toward low-power devices continues, additional microcontrollers may become available that require less power or supply voltage. Also, a more powerful RF energy source could be used in place of the current 5 Watt source.

APPENDIX A

PROTOTYPE TWO SOURCE CODE

```
#include <12C509AG.h>
#fuses INTRC,NOWDT,NOPROTECT,NOMCLR
#use delay(clock=4000000)
#use fast_io(B)

// Internal 4 MHz RC Oscillator Calibration
// Note -- BOARD is command line argument to compiler
#if(BOARD == 4)
   #rom 1023 = {0xC70}
#else
   #rom 1023 = {0xC6C}
#endif

main() {

   int i;
   char boardID[13] = "Board   ID";

   boardID[10] = 0x0D;   // CR
   boardID[11] = 0x0A;   // LF
   boardID[12] = 0x00;   // NULL

   // Note -- BOARD is command line argument to compiler
   #if(BOARD == 1)
      boardID[6] = '1';
   #elif(BOARD == 2)
      boardID[6] = '2';
   #elif(BOARD == 3)
      boardID[6] = '3';
   #elif(BOARD == 4)
      boardID[6] = '4';
   #else
      boardID[6] = 'X';
   #endif

   // Set I/O port direction (GP4 and GP5 outputs)
   set_tris_b(0x0F);

   // Enable RF
   output_high(PIN_B4);
```

```
   // Wait for time slot
   // Each slot is 56 ms
   // Include 2 ms startup time to make sure transmitter on first board is
ready
   delay_ms(2 + (56 * (BOARD - 1)));

   // Output ID via RS232 over RF @ 2400 Baud
   for(i = 0; i < 12; i++) {
      // Mark to space transition - start bit
      output_high(PIN_B5);
      delay_us(409);   // 417 - 8
      // Output eight data bits, no parity
      // Output low = Mark state = Logic "1"
      // Output high = Space state = Logic "0"
      // Bit 0 -- LSB
      if(boardID[i] & 0x01) {
         // Bit 0 is set – output Mark state
         output_low(PIN_B5);
      } else {
         // Bit 0 is cleared – output Space state
         output_high(PIN_B5);
      }
      delay_us(409);
      // Bit 1
      if(boardID[i] & 0x02) {
         output_low(PIN_B5);
      } else {
         output_high(PIN_B5);
      }
      delay_us(409);
      // Bit 2
      if(boardID[i] & 0x04) {
         output_low(PIN_B5);
      } else {
         output_high(PIN_B5);
      }
      delay_us(409);
      // Bit 3
      if(boardID[i] & 0x08) {
         output_low(PIN_B5);
      } else {
         output_high(PIN_B5);
      }
      delay_us(409);
      // Bit 4
      if(boardID[i] & 0x10) {
         output_low(PIN_B5);
      } else {
         output_high(PIN_B5);
      }
      delay_us(409);
      // Bit 5
      if(boardID[i] & 0x20) {
         output_low(PIN_B5);
      } else {
         output_high(PIN_B5);
      }
```

```
      delay_us(409);
      // Bit 6
      if(boardID[i] & 0x40) {
         output_low(PIN_B5);
      } else {
         output_high(PIN_B5);
      }
      delay_us(409);
      // Bit 7 -- MSB
      if(boardID[i] & 0x80) {
         output_low(PIN_B5);
      } else {
         output_high(PIN_B5);
      }
      delay_us(417);
      // Two stop bits
      output_low(PIN_B5);
      delay_us(417);
      output_low(PIN_B5);
      delay_us(408);
   }

   // Disable RF
   output_low(PIN_B4);

}
```

APPENDIX B

PROTOTYPE THREE SOURCE CODE

```
      list      p=12F675         ; list directive to define processor
      #include  <p12f675.inc>    ; processor specific variable definitions

       __CONFIG      _CP_OFF   &   _WDT_OFF   &   _BODEN_ON   &   _PWRTE_OFF   &
_INTRC_OSC_NOCLKOUT & _MCLRE_OFF & _CPD_OFF

;*********************************************************************
;Defines
;*********************************************************************

#define DEV_ID          D'0'        ; Unique device identification number
#define D_SHORT         D'8'        ; Constants for use in transmit routine
#define D_EXACT         D'12'       ; bit time delays
#define D_EXTENDED      D'15'

#define BANK0           0x00
#define BANK1           0x80
#define TrisConfig      B'11101011' ; Initialization values for processor
#define IntconInit      B'10001000' ; configuration registers
#define ANSELInit       B'01010011'
#define ADCON0Init      B'01000001'
#define CMCONInit       B'00000111'
#define _ADEN           D'2'        ; GPIO2 controls thermistor
#define _TXOUT          D'4'        ; GPIO4 controls transmitter

;*********************************************************************
;General Purpose Registers (GPR's)
;*********************************************************************

      cblock      0x20
      WTEMP               ; register used in Interrupt Routine
      STATUSTEMP          ; register used in Interrupt Routine
      PCLATHTEMP          ; register used in Interrupt Routine
      FSRTEMP             ; register used in Interrupt Routine
      intCount            ; interrupt counts
      outer               ; outer delay counter
      inner               ; inner delay counter
      bufHigh             ; buffer for A/D higher byte
      bufLow              ; buffer for A/D lower byte
      bufTransmit         ; buffer for the data byte to be transmitted
```

```
      idx               ; transmit bit index
      dev_id            ; device ID register
      endc

;*********************************************************************
;Reset Vector
;*********************************************************************

      ORG     0x000     ; processor reset vector
      nop               ; required by in-circuit debugger
      goto    Init      ; go to beginning of program

;*********************************************************************
;Interrupt Vector
;*********************************************************************

      ORG   0x004
Isr
      movwf       WTEMP             ;Save off current W register contents
      movf        STATUS,w
      clrf        STATUS            ;Force to page 0
      movwf       STATUSTEMP
      movf        PCLATH,w
      movwf       PCLATHTEMP        ;Save PCLATH
      movf        FSR,w
      movwf       FSRTEMP           ;Save FSR
      BANKSEL     BANK1

;*********************************************************************
;Interrupt Source Checks
;*********************************************************************

Timer0InterruptCheck
      movf  INTCON,w
      andlw 0x20
      btfsc STATUS,Z          ;Is T0IE Set?
      goto  Next1             ;No
      movf  INTCON,w          ;Yes
      andlw 0x04
      btfss STATUS,Z          ;Is TOIF Set?
      goto  Timer0Interrupt   ;Yes

Next1
GPIFInterruptCheck
      movf  INTCON,w
      andlw 0x08
      btfsc STATUS,Z          ;Is GPIE Set?
      goto  Next2             ;No
      movf  INTCON,w          ;Yes
      andlw 0x01
      btfss STATUS,Z          ;Is GPIF Set?
      goto  GPIFInterrupt     ;Yes

Next2
GP2_INT_ExternalInterruptCheck
      movf  INTCON,w
      andlw 0x10
```

```
      btfsc STATUS,Z                    ;Is INTE Set?
      goto  Next3                       ;No
      movf  INTCON,w                    ;Yes
      andlw 0x02
      btfss STATUS,Z                    ;Is INTF Set?
      goto  GP2_INTExternalInterrupt    ;Yes

Next3
PeripheralInterruptCheck
      movf  INTCON,w
      andlw 0x40
      btfsc STATUS,Z          ;Is PEIE Set?
      goto  EndIsr            ;No

Next4
EEIFInterruptCheck
      movf  PIE1,w
      andlw 0x80
      btfsc STATUS,Z          ;Is EEIE Set?
      goto  Next5             ;No
      BANKSEL BANK0           ;Yes
      movf  PIR1,w
      BANKSEL BANK1
      andlw 0x80
      btfss STATUS,Z          ;Is EEIF Set?
      goto  EEPROMInterrupt   ;Yes

Next5
ADIFInterruptCheck
      movf  PIE1,w
      andlw 0x40
      btfsc STATUS,Z                ;Is ADIE Set?
      goto  Next6                   ;No
      BANKSEL BANK0
      movf  PIR1,w
      BANKSEL BANK1
      andlw 0x40
      btfss STATUS,Z                ;Is ADIF Set?
      goto  A_DConverterInterrupt   ;Yes

Next6
CMIFInterruptCheck
      movf  PIE1,w
      andlw 0x08
      btfsc STATUS,Z                ;Is CMIE Set?
      goto  Next7                   ;No
      BANKSEL BANK0                 ;Yes
      movf  PIR1,w
      BANKSEL BANK1
      andlw 0x08
      btfss STATUS,Z                ;Is CMIF Set?
      goto  ComparatorInterrupt     ;Yes

Next7
TMR1IFInterruptCheck
      movf  PIE1,w
      andlw 0x01
```

```
      btfsc STATUS,Z          ;Is TMR1IE Set?
      goto  EndIsr            ;No
      BANKSEL BANK0           ;Yes
      movf  PIR1,w
      BANKSEL BANK1
      andlw 0x01
      btfss STATUS,Z          ;Is TMR1IF Set?
      goto  Timer1Interrupt   ;Yes
      goto  EndIsr            ;No

;*****************************************************************************
;Interrupt Source Code
;*****************************************************************************

Timer0Interrupt
      goto  EndIsr

GPIFInterrupt
      banksel BANK0
      call  DelaySync         ; screen out spurious drop of pin voltage

      btfsc GPIO,0x03         ; Is GP3 high
      goto  EndGPIFInterrupt  ; Do nothing if so

      movf  intCount,w        ; Get the device ID number that is
                              ; currently allowed to transmit
      sublw DEV_ID            ; Check whether the current ID number
      btfss STATUS,Z          ; matches the device ID
      goto  gpie1             ; Increase the counter and return if not

      call  ReadTemp
      setc
      rrf   bufHigh,f
      rrf   bufLow,f
      clrc
      rrf   bufLow,f
      rrf   bufLow,f
      rrf   bufLow,f
      rrf   bufLow,f
      rrf   bufLow,w
      addwf dev_id,w
      call  Transmit
      movf  bufHigh,w
      call  Transmit
      bcf   INTCON,3          ; Disable interrupt because the device has
                              ; completed its mission in this cycle

gpie1
      incf  intCount,f
      btfss intCount,4        ; Restart the count if it is over the device id
                              ; range
      goto  EndGPIFInterrupt
      clrf  intCount
      bcf   INTCON,3          ; Disable interrupt since it is over time

EndGPIFInterrupt
      movf  GPIO,w            ; Clears Mismatch Condition
```

```
      BANKSEL BANK1
      bcf   INTCON,GPIF       ; Clear Interrupt On Pin Change Flag
      goto  EndIsr

GP2_INTExternalInterrupt
      goto  EndIsr

EEPROMInterrupt
      goto  EndIsr

A_DConverterInterrupt
      goto  EndIsr

ComparatorInterrupt
      goto  EndIsr

Timer1Interrupt

EndIsr
      clrf  STATUS            ;Select Bank 0
      movf  FSRTEMP,w
      movwf FSR               ;Restore FSR
      movf  PCLATHTEMP,w
      movwf PCLATH            ;Restore PCLATH
      movf  STATUSTEMP,w
      movwf STATUS            ;Restore STATUS
      swapf WTEMP,f
      swapf WTEMP,w           ;Restore W without corrupting STATUS bits
      retfie                  ;Return from interrupt

;*****************************************************************************
;Initialization
;*****************************************************************************

Init
      call  0x3FF       ; retrieve factory calibration value
                        ; comment instruction if using simulator, ICD2, or
                        ; ICE2000
      BANKSEL BANK1     ; BANK1
      movwf OSCCAL      ; update register with factory cal value
      movlw TrisConfig  ; set direction so that pins 3 and 5 (GP4 and GP2)
                        ; are outputs
      movwf TRISIO      ; all others are inputs (high-z)
      movlw IntconInit  ; configure interrupt control register
      movwf INTCON      ; so that IOC on GP3 is enabled
      movlw ANSELInit   ; configure GP0 and GP1 (A/D Vref) as analog input
      movwf ANSEL       ; Fosc/16 for A/D clock
      clrf  VRCON       ; comparator Vref off
      bsf   IOCB,3      ; interrupt on pin change for GP3
      BANKSEL BANK0     ; change back to PORT memory bank
      movlw CMCONInit   ; configure comparator inputs as digital I/O
      movwf CMCON
      movlw ADCON0Init  ; configure ADCON0 so the output is left justified
      movwf ADCON0      ; and the voltage reference is Vref

      bcf   GPIO,_ADEN  ; clear GPIO ports
      bcf   GPIO,_TXOUT ; clear GPIO ports
```

```
      clrf  intCount    ; initialize the interrupt counter
      movlw DEV_ID
      movwf dev_id      ; put the device ID to the dev_id register
      clrc              ; clear the carrier flag
      rlf   dev_id,1    ; shift the ID to bits 6:3
      rlf   dev_id,1
      rlf   dev_id,1

;*****************************************************************************
;Main
;*****************************************************************************

Main
      bcf   GPIO,_TXOUT       ; clear transmit port
      nop
      sleep
      nop
      goto  Main

;*****************************************************************************
;Subroutines & Functions
;*****************************************************************************

;*********************************************************
;ReadTemp subroutine
;Get data from thermistor to send out async port into w
;INPUT: none
;Output: W
;VARIABLES: idx(counter)
;*********************************************************

ReadTemp
            banksel BANK0
            bsf   GPIO,_ADEN  ; turn on thermistor
            call  DelayAD
            bsf   ADCON0,1    ; turn on A/D
waitlp:     btfsc ADCON0,1
            goto  waitlp
            bcf   GPIO,_ADEN  ; turn off thermistor

            movf  ADRESH,0    ; move the higher byte to w register
            movwf bufHigh     ; move the result to buffer

            banksel BANK1
            movf  ADRESL,0    ; move the lower byte to buffer
            banksel BANK0
            movwf bufLow

            return

;*****************************************************************************
; Transmit
;*****************************************************************************

Transmit
            banksel BANK0
```

```
            movwf bufTransmit   ; Move byte to transmit from w into
                                ; bufTransmit
            movlw d'9'          ; Need to send 9 bits (not counting stop bit)
            movwf idx
            clrc                ; Set the carry ("previous bit") to zero --
                                ; start bit
            rrf   bufTransmit,1 ; Shift the start bit to the current bit
                                ; position (7)

                                ; Transmit start bit '0'
            btfsc bufTransmit,6 ; This bit is a '0' -- is the next bit a '1'?
            goto  TxmZero
            bsf   GPIO,_TXOUT   ; The next bit is a zero -- transmit a '0'
                                ; with exact delay
            call  DelayExact
            goto  TxmNext

TxmStart
            btfsc bufTransmit,7 ; Is the current data bit '0'?
            goto  TxmOne        ; Goto transmit a '1'
            btfsc bufTransmit,6 ; This bit is a '0' -- is the next bit a '1'?
            goto  TxmZero
            nop
            bsf   GPIO,_TXOUT   ; The next bit is a zero -- transmit a '0'
                                ; with exact delay
            call  DelayExact
            goto  TxmNext

TxmZero                         ; The current bit is a 0 and the next is a 1
                                ; There is a delay switching the transmitter
                                ; off -- Compensate by using a smaller delay
                                ; this time and a larger delay the next
            bsf   GPIO,_TXOUT
            call  DelayShort
            goto  TxmNext

TxmOne
            skpc                    ; Was previous bit a zero?
            goto  TxmOneExtended    ; Yes -- needs special processing
            bcf   GPIO,_TXOUT       ; Transmit a '1'
            call  DelayExact
            goto  TxmNext

TxmOneExtended                  ; The previous bit was a 0 and this is a 1
                                ; There is a delay switching the transmitter
                                ; off -- Compensate by using a larger delay
                                ; this time (a smaller delay was used last
                                ; time)
            bcf   GPIO,_TXOUT
            call  DelayExtended
            goto  TxmNext

TxmNext
            rlf         bufTransmit,1     ; current bit -> carry
            decfsz      idx,1             ; decrease the bit count
            goto        TxmStart
            bcf         GPIO,_TXOUT       ; send the stop bit of '1'
```

```
            call        DelayExact
            return

;*****************************************************************************
; DelayShort
;*****************************************************************************

DelayShort
      movlw       D_SHORT
      movwf       inner
DB0
      decfsz      inner,f     ; loop countdown
      goto        DB0
      nop
      nop
      return

;*****************************************************************************
; DelayExact
;*****************************************************************************

DelayExact
      movlw       D_EXACT
      movwf       inner
DB1
      decfsz      inner,f     ; loop countdown
      goto        DB1
      return

;*****************************************************************************
; DelayExtended
;*****************************************************************************

DelayExtended
      movlw       D_EXTENDED
      movwf       inner
DB2
      decfsz      inner,f     ; loop countdown
      goto        DB2
      return

;*****************************************************************************
; DelayAD: delay for the hold capacitor in A/D to charge up
;*****************************************************************************

DelayAD
      movlw       D'20'
      movwf       inner
DBAD
      decfsz      inner,f     ; loop countdown
      goto        DBAD
      return

;*****************************************************************************
; DelaySync: delay to make sure it is the synchronization bit
;*****************************************************************************
```

```
DelaySync
      movlw       D'5'
      movwf       inner
DS0
      decfsz      inner,f     ; loop countdown
      goto        DS0
      return



      END                     ; directive 'end of program'
```

BIBLIOGRAPHY

1. M. H. Mickle, M. Lovell, L. Mats, L. Neureuter, and D. Gorodetsky, "Energy Harvesting, Profiles, and Potential Sources," *International Journal of Parallel and Distributed Systems and Networks,* vol. 4, pp. 150-160, 2001.

2. M. H. Mickle and James T. Cain, "Passive Wireless Sensors for Spacecraft Applications," University of Pittsburgh, 2005.

3. Microchip Technology Inc., *rfPIC12C509AG / 509AF Data Sheet,* 2001.

4. ECS Inc. International, *CSM-7 SMD Crystal for TX/RX Chipset*, 2002.

5. Microchip Technology Inc., Appl. Note 846, pp. 1-8.

6. Custom Computer Services, Inc., *PICmicro® MCU C Compiler Reference Manual,* 2002.

7. M. Mi and M. H. Mickle, "Annealing Approach to the Impedance Matching of Antennas for RFID Tags," Manuscript submitted to IEEE Microwave and Wireless Components Letters, University of Pittsburgh.

8. Microchip Technology Inc., *PIC12F629 / 675 Data Sheet,* 2003.

9. E. F. Knott, J. F. Shaeffer, and M. T. Tuley, *Radar Cross Section*. Boston: Artech House, 1993.

10. Microchip Technology Inc., *PIC16F870 / 871 Data Sheet,* 2003.

11. Panasonic, *Multilayer Chip NTC Thermistors (ERTJ)*, Rev.02/04.