ENERGY MONITOR

A Design Project Report

Presented to the Engineering Division of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements of the Degree of

Master of Engineering (Electrical)

by

Xi Guo

Project Advisor: Bruce Land

Degree Date: May 2009

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: Energy Monitor

Author: Xi Guo

Abstract:

The system detailed in this document was originally designed to monitor and control the energy usage of household electric appliances through wall-sockets and a separate handheld interface device with near field communications with 13.56 MHz carrier frequency. After researching and looking for similar project, each individual system (e.g. power sensing circuit, near field communication) of the project was designed, built, successfully implemented and tested. However, due to time constraints and lack of testing, the assembly of the complete system was not finished and could not be fully tested. The ultimate goal for this project is to fully design and build a working prototype of the power monitoring/controlling system that is safe, easy to use, accurate, as well as inexpensive to make with readily available parts. Even though the project failed to provide a working prototype of the original design, the experience learned working with the communication system as well as the final design and implementation could prove to be useful in future projects.

Report Approved by

Project Advisor: _____ Date: _____

Introduction/Overview:

For most families, the only opportunity gauge the power usage is the monthly power bill for the whole house, this means that it is difficult to determine which appliance(s) use the most power during specific times. Having the ability to monitor the power usage at wall-socket level can changes this and allow people to compare and micromanage the usage of their appliances. Not only will this lower the overall power bill with minimal effort, but at the same time it will also reduce the demand for power and help save the environment.

Individual aspects that make up this project has already been well-researched and understood. For example, power monitoring at fine grain (socket) level has already been implemented and even been produced commercially in forms of product such as Kill-A-Watt or Watts-up. An issue with these products is that they are designed to be standalone after-market components, meaning they are designed to be used with existing sockets, and that each individual socket contains its own user interface. This means not only will these units be always visible, but there is no synergy in having multiple units of the power meter (it doesn't make sense to have a LCD display and control panel for each unit).

This design project will attempt to change this by separating the power sensing module and a user interface module. This makes monitoring each of the individual power monitoring sockets less expensive by using the same user interface module, and as well as allow the unit to be hidden inside the wall with the rest of the socket circuitry. In addition, since the information is transmitted wirelessly to the user, there is never a direction connection between the user and ac main line, making the system safer by design.

Because now that there is a layer of dry wall between the power sensing module and the user, the power usage information has to be relayed wirelessly. Although other wireless communication methods exist for microcontrollers (MCUs), near field communication is the most appropriate one for this application, due to its broadcast range complexity. Near field communication (NFC) is a short range wireless technology allows two devices to communicate via high frequency magnetic coupling without requiring a line of sight. NFC works in the same way as RFID (radio frequency identification), which is already very prevalent in our lives in today's world (e.g. Cornell's student ID contactless entry system to dorms). The frequency of the carrier signal that will be used in this project is 13.56 MHz. As the relatively higher frequency (compared to 125 kHz) corresponds to smaller antennas, the smaller antennas will allow help reduce the footprint of the communication circuitry and minimize the size of the modules.

In summary, the project should meet the following design goals:

- Wirelessly transmit data from power sensing circuitry to a separate handheld display with 13.56 MHz carrier frequency at a reasonable rate
- Electrically isolated user interface, microcontroller, and power sensing circuit
- Circuits built with readily available parts to reduce cost

Related Works:

The power sensing circuitry and some of the code of this project will be heavily borrowed from a Cornell ECE476 class final project, "PowerBox: The Safe AC Power Meter". (Jao, C.; Guo, X.;, 2008) In the project as shown in the figure below, the power meter uses a small power resistor to detect the current draw through the socket, and a voltage divider circuit to detect the actual voltage across the AC live and neutral line. These values are then opto-isolated, linearly

amplified and fed to the microcontroller's ADC. A relay was used to optionally control the operation of the device under measurement.

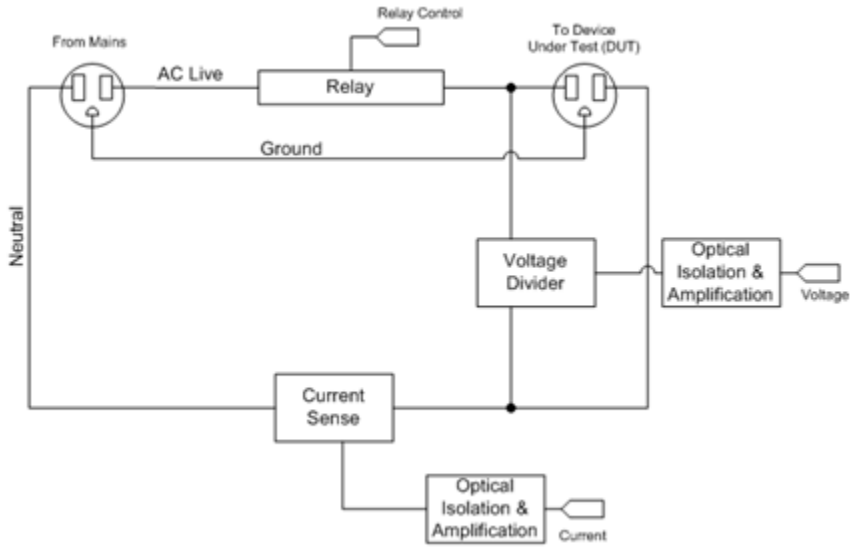


Figure 1: PowerBox system block diagram. Note: the socket on the right has AC live (short blade) on the left (Jao, C.; Guo, X.;, 2008)

There have been many published RFID or NFC implementations. One of the more pertinent project based on microcontrollers is the "Minimum Mass Wireless Coupler". (Cappel, 2005) The NFC system described in the paper uses a 181.818 kHz carrier frequency with on-off keying (OOK) to transmit data from RS-232 to 2 line by 16 character liquid crystal display (LCD). It uses the microcontroller (AMega8) to generate the generate the carrier and modulate the signal with the onboard tri-state driver, which is then fed to an inductor and capacitor in parallel. The receiving circuit involves the same inductor and capacitor fed into microcontroller's analog comparator, through which the can detect the presence of the carrier. To achieve the desired resonant frequency, the system uses a 5.5 cm diameter wire loops with 14 turns and 33nf capacitors.

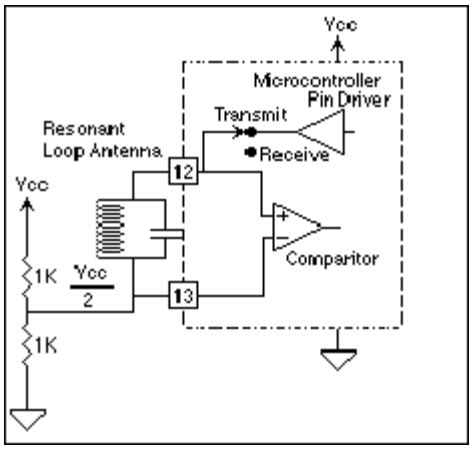


Figure 2: Circuit diagram for the Minimum Mass Wireless Coupler (Cappel, 2005)

A second microcontroller project that uses RFID frequency is another Cornell ECE476 final project, “Proximity Security System”, (Ross, Craig; Goto, Ricardo, 2006) which detailed their design process towards reading information from commercial contactless ID cards. This system is different from the Minimum Mass Wireless Coupler in that it uses 125 kHz carrier frequency with Frequency Shift Keying (FSK) at 12.5 kHz and 15.625 kHz. Similarly though, as one can see in the figure below, the reader also generates the carrier frequency from the microcontroller directly, passing only through some current amplifiers and an inductive choke to reduce the upper harmonics. The RFID tag will receive the signal and reemitted a frequency modulated version with frequency dependent on the data encoded in the card. This signal is then rectified to find the signal envelop and filtered to the appropriate frequency. One point of interest in their design is that they used a Johnson (decade) counter to help reduce the amount of interaction needed with the MCU.

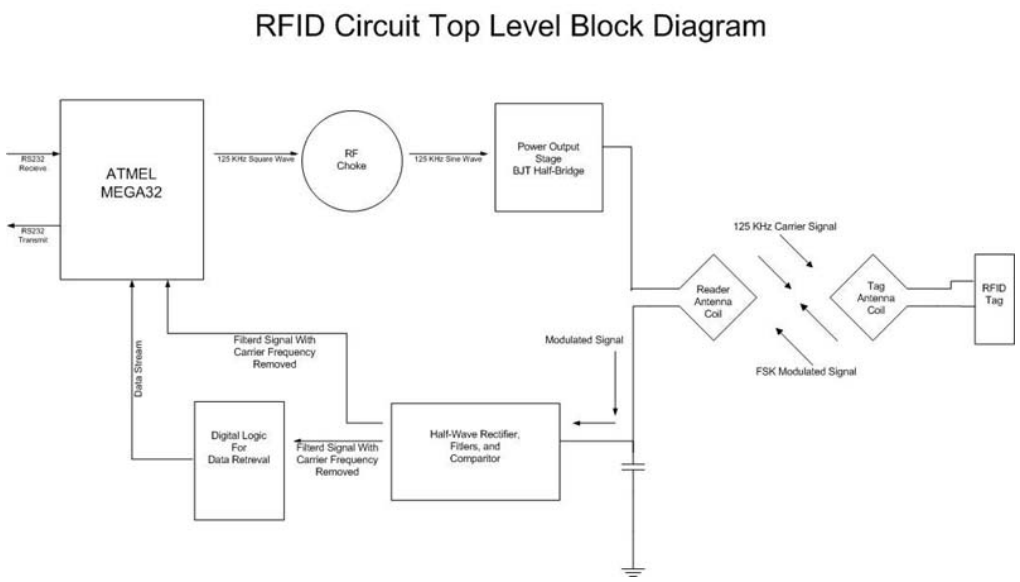


Figure 3: Top Level Design Diagram for RFID Reader (Ross, Craig; Goto, Ricardo, 2006)

Since the power sensing circuit will be almost all borrowed directly from the PowerBox project, the core of this project is designing and building the necessary hardware and communication protocol to allow the modules to communicate within the desired 13.56 MHz frequency band. Since this is at a much higher frequency than the projects described above, the guides and instructions given above are not directly helpful, but are only good as references.

Design and Implementation Process:

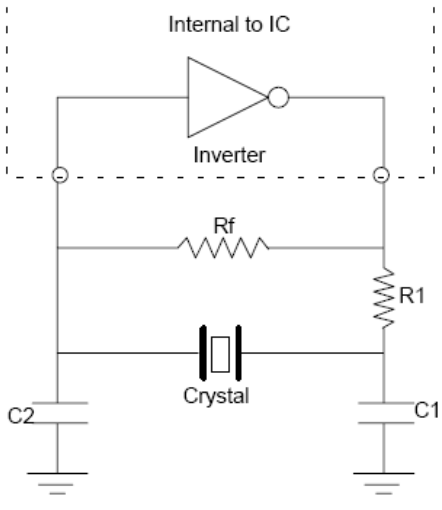
This part of the report will describe in detail both the process necessary to eventually develop the design as well as some of the possible solutions that did not entirely work.

The problem involved with communication at 13.56 MHz can be separated into the different areas: clock generation, signal modulation, antenna/coupler design, data retrieving, and

communication protocol. Each of which required the completion of the previous areas to be tested effectively.

Clock generations

In both of the previous projects described above, the carrier frequency was relatively low (125, 181 kHz) and they were generated directly from the MCU. Because the target carrier frequency for this project is different from but on the same order of magnitude as the MCU clock frequency (16 MHz), MCU cannot be used to generate the waveform directly, but instead an oscillator must be used. The bigger problem is that there are no oscillators at the exact frequency on the site of major online electrical component retailers, but instead only surface mount crystals are sold. Since part of the goal is to design with only readily available components, I decided to build oscillators from the crystals. Some research led to an application note with basic crystal oscillator circuit design. (Crystal Oscillator Circuit Design, 1997) As seen in the figure below, an active inverter element biased at Vcc/2 was used to drive the crystal 180 degrees out of phase with itself. Originally I planned the 74LS04 inverter package.



$$C_L = \frac{C_1 * C_2}{C_1 + C_2} + C_S$$

Figure 4: Crystal oscillator circuit and equation to calculate load capacitance

C1 and C2 were determined with the equation by working backwards (assuming C1 = C2, Cs is stray capacitance from circuit board, ~5pF) from the crystal's rated capacitance, C_L, 18 pF. C1 = C2 = (18-5)*2 = 26 pF. Capacitors with capacitance of 22 pF , 1M Ohm resistor for Rf were used. R1 wasn't necessary in my case for the oscillator.

Initially the output stalled at a DC voltage. A talk with my advisor, Dr. Bruce Land, later reveals that the inverters in 74LS04 are not CMOS, thus they do not have the steep gain at Vcc/2 on their input-output graph. After replacing the inverter package with 74C04, the oscillator worked as expected. The output of oscillator was then fed through another inverter to be further amplified.

Signal modulation

Between two of the more common modulation schemes were introduced in the related work sections, OOK is more relevant to this project since the transmitter has its own power supply and that it would be unnecessary to have two more crystals/oscillators. Since the MCU no longer handles the clock generation in this project, the on-chip tri-state buffer cannot be used and external chip has to be used. DM74125N was used, and it worked as expected, although it seems a little wasteful to only use one of the four gates in the package.

Antenna/coupler design

Both of the related works referenced described the processes of designing the inductor to use as an antenna for transmitting and receiving. The equation to compute the necessary inductance is $f_0 = \frac{1}{2\pi\sqrt{LC}}$, where f_0 is the desired carrier frequency, L and C are the values for the inductor and

capacitor of the antenna being driven at frequency. A capacitance value of 100 pF was chosen, and the corresponding inductance was then $L = \frac{13.56e6*2\pi}{1e-10} = 1.38\ \mu H$.

Another formula was use to approximate the number of turns $L \approx N^2 R u_0 [\ln\left(\frac{8R}{a}\right) - 2]$. With radius of loop, R = 0.02 m, radius of the wire, a= 0.000203 m, approximate 4 turns will result in the desired inductance. Although the circuit (somewhat) worked, the range required to achieve a reasonable output peak-peak voltage is extremely short, this makes the signal processing and data retrieval much more challenging.

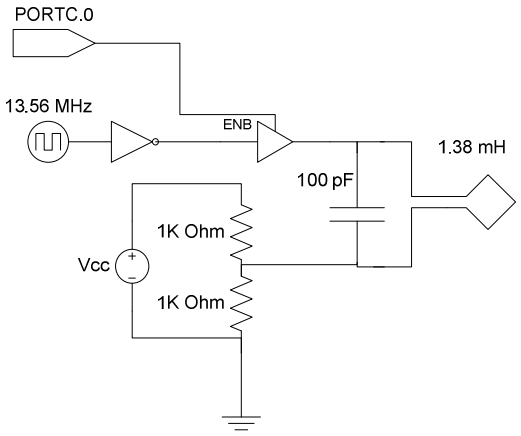


Figure 5: Schematic of final transmission hardware

Putting everything together, the figure above shows the entire transmission circuits implemented. The modulation of the wave is controlled by a microcontroller pin (C0 in this case).

Signal filtering/detection

At first, a modified version of the approach described in the Minimum Mass Wireless Coupler was considered: using two resistors to bias one end of the antenna at Vcc/2, an analog comparator can be used to detect the smallest change in the sign of the received. Because an

analog comparator with gain-bandwidth product higher than 1MHz could not be acquired on a timely basis, using fast op amp was not a feasible solution for this project.

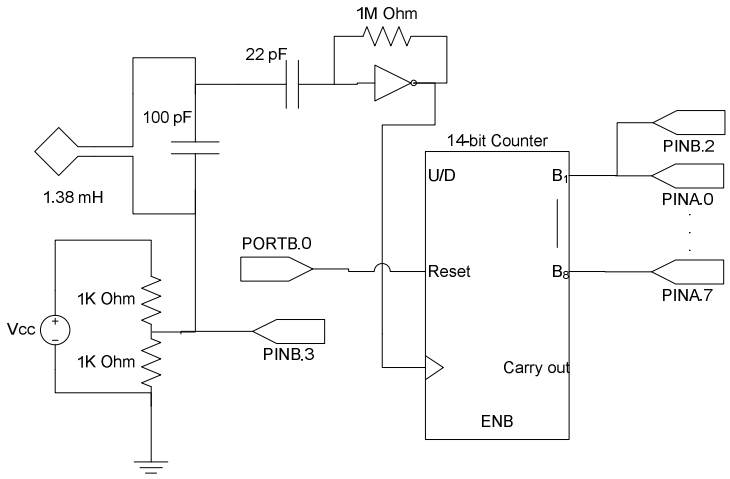


Figure 6: Schematic of the final receiving hardware

So instead of using op amps boost the voltage level, I then attempted simply replace them with inverters, thinking that that the voltage will be stable enough to remain around Vcc/2 and be amplified. However, that didn't seem to be the case and the signal became weaker. The input voltage of the inverter had to be biased at Vcc/2, so instead of biasing the antenna with voltage divider, a 22 pF capacitor was placed in series between the antenna output and the inverter input to block the DC voltage and passing only AC, and a 1 M Ohm resistor was used to connect the input and output of the inverter to bias the input as shown in the figure above, similar to the setup seen for the crystal oscillator. This yielded logic level signal, but a problem still exists that the frequency of the signal is too high to be analyzed by the MCU. The use of external counter in the RFID card reader project prompted me to look for the 14-stage binary counter. By connecting the signal to its clock input, the counter acts as a clock divider so that the result may be polled. The 8 high bits outputs of the counters are connected to the MCU as input (pin A in the case shown in the figure above). An output pin of the MCU is used to clear the counter at the end of every

sampling period (baud period). Pin B3 of the MCU is connected to the voltage divided Vcc/2, and a lower order bit of the counter is also connected to Pin B2 of the MCU to use to trigger the analog comparator to provide a way to detect the initial edge of the pulse that is more time accurate and robust than the polling method.

With the given hardware setup, 2 way communication is possible as the antenna portion of the schematic are the same for both transmission and reception, and also that the I/O resource of the MCU is not conflicting. But due to the nature of original design to have dedicated transmitter and receiver, 2 way communication was not implemented.

Communication protocol

For convenience, using the divide by 8 prescaler and a clear-on-match value of 124, I decided to run the signal at a baud rate of 16 MHz/1000 = 16 kHz. To allow the receiver to properly latch on the signal, a long period of high and low is always sent before the data. A packet of data is shown in the figure below: the black regions indicate signals that will always be sent with the packet regardless of the data, and the gray region indicate variable signal depending on data. The figure below shows sample data of 0, 1, 0, 0, 1, 1, 0, 0, 1… The first high sync pulse will be used to signal the immediate arrival of data, then the low sync pulse allows the receiver to be ready and prepare for the first uptick to latch on, which will be captured through the analog comparator. Upon capturing the first uptick, the MCU will switch into periodic polling mode, sampling to see if the counter has reached a threshold within the baud period of 1/16kHz = 62.5 microseconds and clearing immediately after to prepare for the next sample.

Figure 7: A packet of data (gray) with preamble sync pulses (black)

The implementation of this transmission protocol is done with a state machine inside a timed interrupt (at baud frequency). The graph below shows the state required to generate the signal packet with the preamble pulses. In the state machine, High sync and Low sync both holds the output at the two levels for some integer multiple of the baud period. At the end of the Low sync, the output is turned high to generate the leading bit of the packet used to sync with the receiver. The transmitter than processes through the data, shifting one bit at a time starting with MSB until all of the data has been transmitted. Then it idles to wait on the next command.

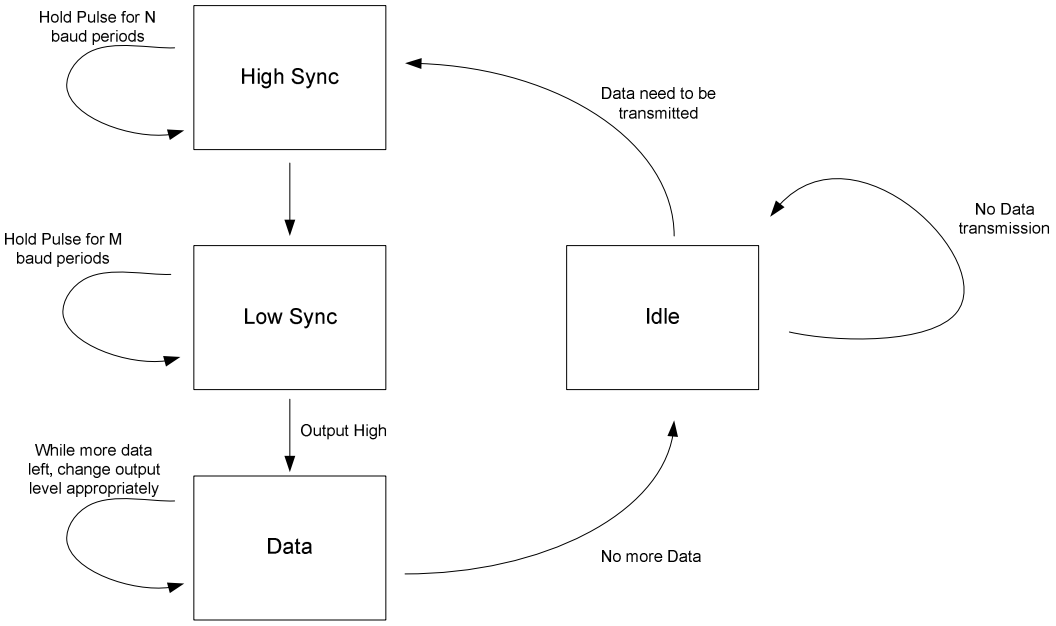


Figure 8: Transmission state transition diagram

The receiving and data decode state machine is very similar. The figure on the next page shows the state machine required to receive and decode the signal. Because the receiver should always

be expecting incoming data, an idle state does not exist, but instead it's always looking for the initial high pulse lasting N baud periods. After seeing the pulse, it then waits for M baud cycles to be sure that the packet is now in the low sync region and turns on the analog comparator. A self-resetting mechanism was added to rest the state if leading bit of the packet is not seen for the duration of the low sync. After the first bit has been detected, the receiver moves to data mode, sampling and then resetting the counter every baud period. For each of samples, the count was compared to a general threshold to determine if it was logic high or low.

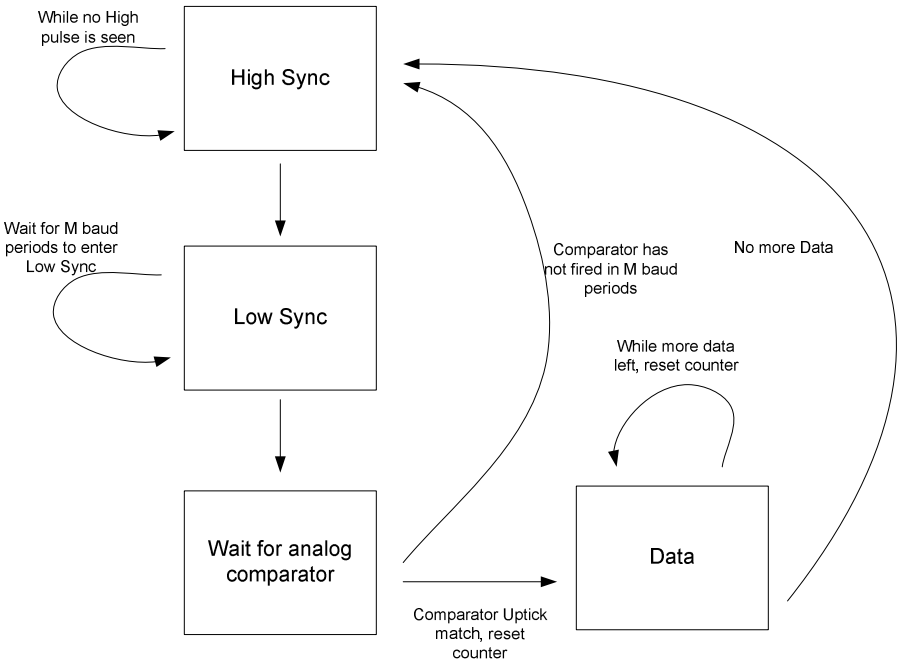


Figure 9: Receiving state transition diagram

Due to various reasons (computation delay, counter delay) the data didn't always align exactly with the sampling, "bleeding" of the data occurred where a single bit of low among a packet of highs can sometime cause the neighboring bits to appear as low also. This is because the single period of low can now be detected by two samples, and with less than the full count, neither of

the samples is above the threshold for logic high, and thus two low appears on the receiver end. Reducing the threshold provoked bleeding in the other direction (two highs appearing from source data of a single high bit among lows). This prompted me to use a relative method to determine the logical value of the samples, and Manchester code was used. In Manchester coding, a single bit of data value is now represented by the transition between a pair of bits. The IEEE standard designate that a low-to-high transition in the bit stream represents a logical high, and that a high-to-low transition represents a logical low. For the packet stream in the figure below, the corresponding data stream is 1, 1, 0, 0, 0…

Figure 10: Beginning of packet of data encoded in Manchester code

Using Manchester coding means that the transmission stream will be twice as long, but now instead of using a single threshold to check for the bit value, the count values of a pair of data can be taken and compared to determine the logic value. In addition, because on average every other bit in the data portion of the packet is a 0 and the other is a 1, it's no longer possible to have a data pattern that "looked" like sync pulses (a series of 1's and then a series of 0's).

Result:

The finished communication system can transmit and receive at the planned baud rate (16 kHz) with no perceivable error, however at very limited range. A proof-of-concept demonstration was setup and tested the system. The demonstration uses two STK-500 programming board, with one attached to a set of push buttons and the transmitter, and the other connected to a set of LED and the receiver. When a collection of buttons are pushed on the transmitter, the corresponding LED

on the receiver board will be turned on with almost unperceivable delay. The Powerbox code was ported to Mega644 from Mega32, from Codevision to GCC but this was not tested due to time constraint, so the NFC system could not be integrated unto the module and be tested.

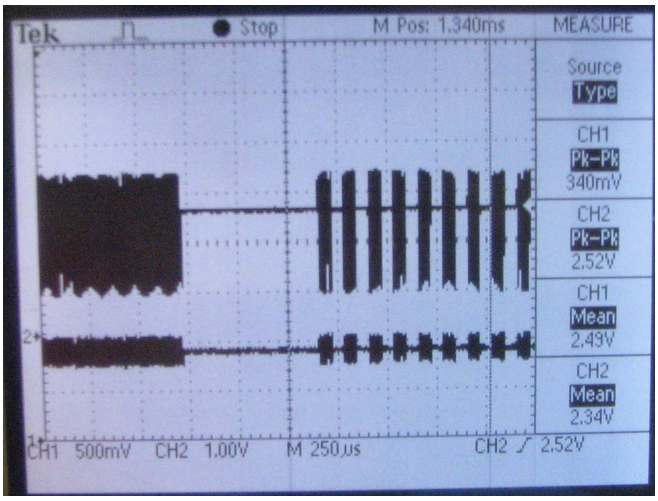


Figure 11: Signal on both side of communication channel (top is input, bottom is output)

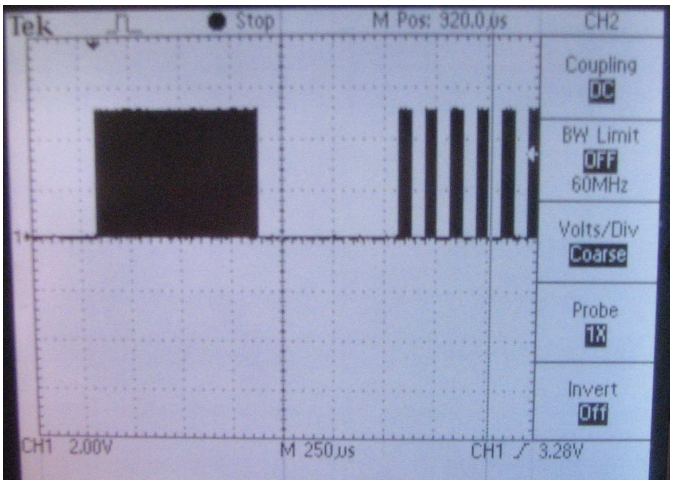


Figure 12: Received signal from counter. Note the pulses preceding the data packet

Figure 11 above shows the signal across the transmitting and received antennas with packets in Manchester coding. Figure 12 shows the received signal after passing through the counter. After boosting the signal to logic level, MCU can then be used to poll the data as needed.

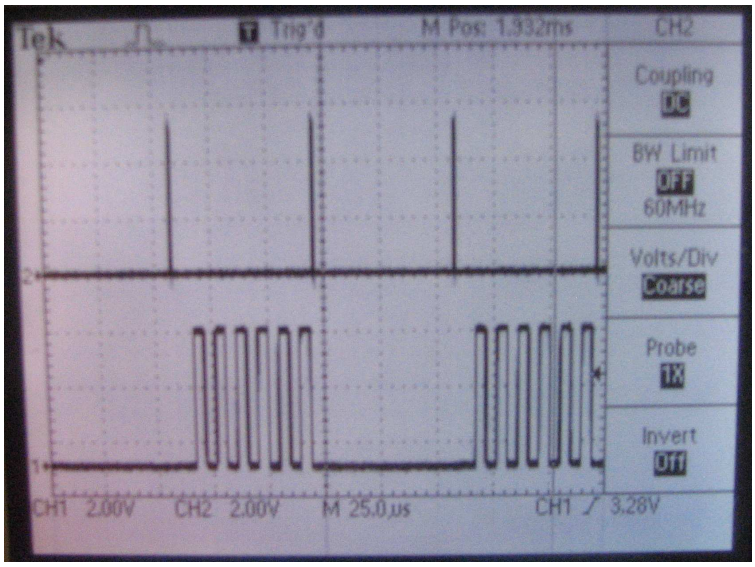


Figure 13: Counter signal with reset_counter signal on top

Figure 13 shows the sampling period by the receiver with the reset_counter pulse on top, and the counter output on bottom. Notice that the reset_counter pulse doesn't line up exactly with the signal pulse, which is the reason that Manchester coding is required to work around this error in timing.

Conclusion:

Overall the project did accomplish design and implementation of the core communication system with readily available components. However, it is rather unsatisfactory to not have the system integrated into the main application and to have the communication system only working on very close range. I believe the main difficulty with this project was the use of 13.56 MHz carrier frequency. Seeing that the only 13.56 MHz devices available in retail are fully assembled commercial reader or programmable systems, I do not consider this experience of learning and building everything from scratch a failure. With a basic NFC system completed, this should allow future projects on the same topic to have some starting information or at the very least not

run into the same obstacles. Given more time and resources, I would like to extend the usable range of the system by a combination of improving antenna, the filter system, or implementing ECC within the transmission. Also, it would be interesting to fully implement bidirectional communication with an added layer of handshake protocols.

References:

Cappel, D. (2005, March). *Minimum Mass Wireless Coupler*. Retrieved March 2009, from Dick Cappel's projects pages: http://cappels.org/dproj/minmassrf/Min_Mass_Wireless_Coupler.html

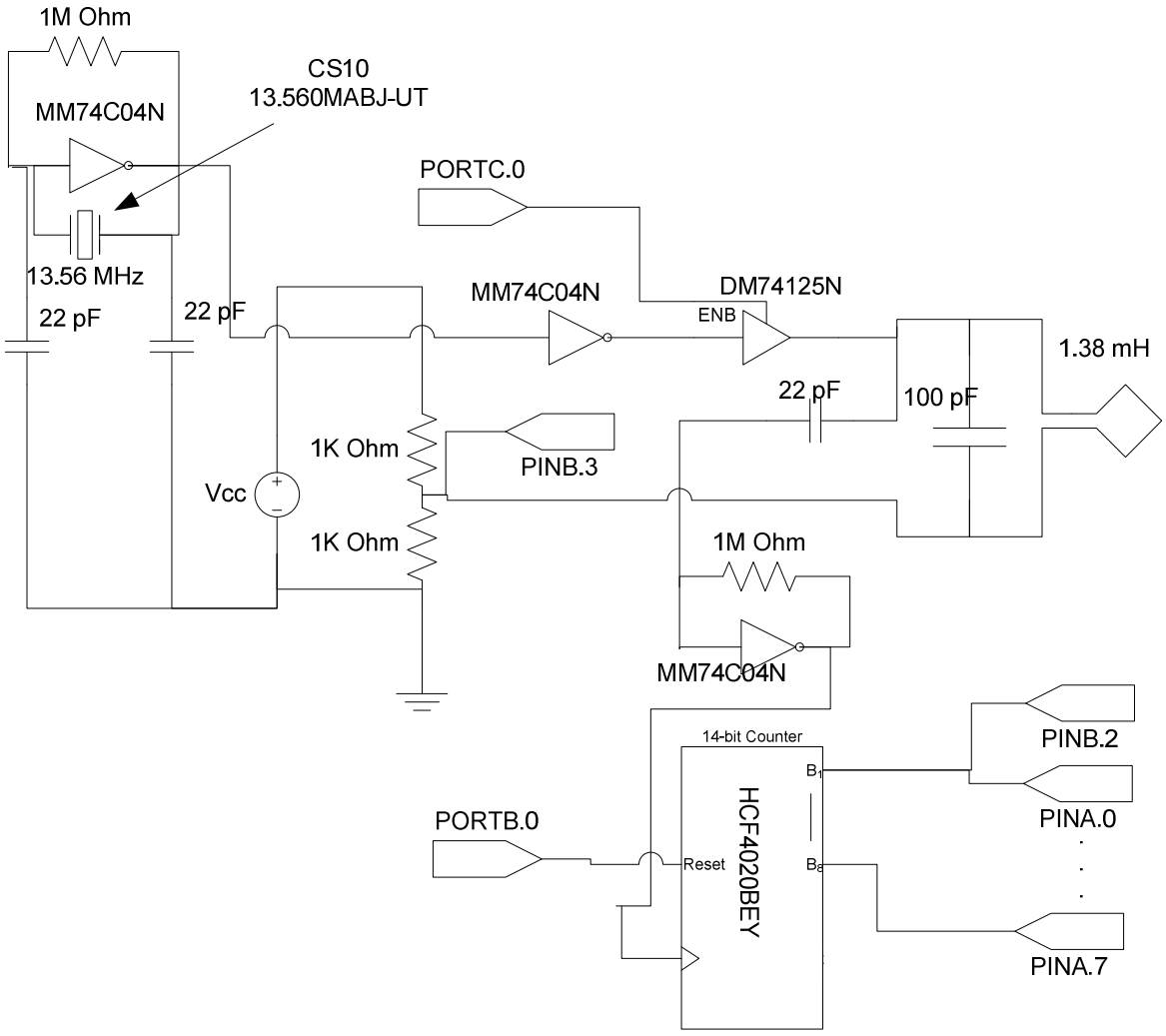
Crystal Oscillator Circuit Design. (1997). Retrieved from mxcom: http://www.eetkorea.com/ARTICLES/2001SEP/2001SEP06_AMD_AN.PDF

Jao, C.; Guo, X.;. (2008, May). *PowerBox: The Safe AC Power Meter*. Retrieved from Cornell ECE476 course site: http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2008/cj72_xg37/cj72_xg37/index.html

Ross, Craig; Goto, Ricardo. (2006, May). *Proximity Security System*. Retrieved from Cornell ECE476 course site: http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2006/cjr37/Website/index.htm

Appendix A:

Full Schematic



Appendix B:

Transmitter Demo Code

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <util/delay.h>


#define begin {
#define end }

#define mode_highsync 1
#define mode_lowsync 2
#define mode_wait 3
#define mode_data 4


void initialize(void); //all the usual mcu stuff

volatile unsigned char outreg, count;
volatile int time1, time2 ;
volatile char mode, bit;


//*********************************************************
//timer 0 overflow ISR
ISR (TIMER0_COMPA_vect)
begin

  if (mode == mode_highsync) {
      if (time1>0)      {

            --time1;
      }
      else {
            mode = mode_lowsync;
            PORTC = 0x01;
            time2 = 10;
      }
  }
  else if (mode == mode_lowsync) {
      if (time2>0)      {
            --time2;
      }
      else {
            PORTC = 0x00;
            count = 0;
            outreg = PINA;
            mode = mode_data;
            bit = 0x00;
      }
  }
  else if (mode == mode_data) {

      if ((outreg & 0x80) == 0x00)
            PORTC = bit;
      else
            PORTC = (0x01 ^ bit);
      if (bit == 0x01) {
            outreg = outreg << 1;
```

```
            bit = 0x00;
      }
      else bit = 0x01;
      if (++count > 16) {
            mode = mode_wait;
            PORTC = 0x01;
            time1 = 100;
      }

  }
  else if (mode == mode_wait) {
      if (time1>0)      {
            --time1;
      }
      else {
            PORTC = 0x00;
            mode = mode_highsync;
            time1 = 12;
      }
  }
end


//*********************************************************
//Entry point and task scheduler loop
int main(void)
begin
  initialize();

  // main task scheduler loop
  while(1)
  begin

  end
end



//*********************************************************
//Set it all up
void initialize(void)
begin

  TIMSK0= (1<<OCIE0A);  //turn on timer 0 cmp match ISR
  OCR0A = 124;          //set the compare re to 125 time ticks
  //set prescalar to divide by 8
  TCCR0B= 2; //0b00000010;
  // turn on clear-on-match
  TCCR0A= (1<<WGM01) ;

  DDRB = 0 ;

  DDRC = 0xff;
  DDRA = 0x00;

  time1=10;
  mode = mode_highsync;

  //crank up the ISRs
  sei();

end
//=================================================
```

Appendix C:

Receiver Demo Code

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#define F_CPU 16000000UL  // 1 MHz
#include <util/delay.h>
#include <string.h>
#include <stdlib.h>
#include <avr/pgmspace.h>




#define begin {
#define end }

#define mode_highsync 1
#define mode_lowsync 2
#define mode_wait 3
#define mode_data 4


//the task subroutine
void task1(void);
void task2(void);

void initialize(void); //all the usual mcu stuff

//timeout counter for task1
volatile unsigned char led, inreg, count;

// timer 1 capture variables for computing sq wave period
volatile unsigned int T1capture, lastT1capture, period;
volatile int time1, time2 ;
volatile unsigned char mode, bit, last;

// polling variables for computing sq wave period (low accuracy)
unsigned int periodPoll, T1poll, lastT1poll ;

//comparator output bit
char ACObit, lastACObit ;

//*********************************************************
//timer 0 overflow ISR
ISR (TIMER0_COMPA_vect)
begin

  if (mode == mode_highsync) {
      if (time1>0)      {
            --time1;
            if (PINA >= 0x40) {
                  time2 = 10;
                  mode = mode_lowsync;

            }

      }
      else {
```

```
            time1 = 10;
            PORTB = 0x01;

            _delay_us(0.2);
            PORTB = 0x00;
      }
  }
  else if (mode == mode_lowsync) {
      if (time2>0)      {
            --time2;
      }
      else {
            PORTB = 0x01;
            _delay_us(0.2);
            PORTB = 0x00;
            ACSR |= (1<<ACIE);
            inreg = 0;
            count = 0;
            mode = mode_wait;
            time2 = 20;
      }
  }
  else if (mode == mode_data) {
      if (bit == 0x00) {
            last = PINA;
            bit = 0x01;
      }
      else {
            bit = 0x00;
            if (last < PINA)
                  inreg = (inreg << 1) | 1;
            else if (last > PINA)
                  inreg = (inreg << 1);

      }

      PORTB = 0x01;
      _delay_us(0.2);
      PORTB = 0x00;
      if (++count > 16) {
            led = inreg;
            mode = mode_highsync;
            time1 = 10;
      }

  }

  else if (mode == mode_wait) {
      if (time2>0)      {
            --time2;
      }
      else {
            ACSR &= ~(1<<ACIE);
            PORTB = 0x01;
            _delay_us(0.2);
            PORTB = 0x00;
            time1 = 10;
            mode = mode_highsync;
      }
  }
end

//*********************************************************
// timer 1 capture ISR
```

```
ISR (ANALOG_COMP_vect)
begin
      ACSR &= ~(1<<ACIE);

      _delay_us(53);

      PORTB = 0x01;
      _delay_us(0.2);
      PORTB = 0x00;
      bit = 0x01;
      mode = mode_data;
      TCNT0 = 10;

end


//************************************************************
//Entry point and task scheduler loop
int main(void)
begin
  initialize();
  // main task scheduler loop
  while(1)
  begin
      PORTD = led;
  end
end
//************************************************************
//Set it all up
void initialize(void)
begin

  led = 0x00;
  //set up timer 0 for 1 mSec timebase
  TIMSK0= (1<<OCIE0A);  //turn on timer 0 cmp match ISR
  OCR0A = 124;          //set the compare re to 125 time ticks
  //set prescalar to divide by 8
  TCCR0B= 2; //0b00000010;
  // turn on clear-on-match
  TCCR0A= (1<<WGM01) ;

  //capture an edge on analog comparator pin B.3
  // Set capture to positive edge, full counting rate
  // Set analog comp to connect to timer capture input
  // and turn on the band gap reference on the positive input
  ACSR = 3;
  // Comparator negative input is B.3
  DDRB = 0x03 ;


  DDRD = 0xff;
  DDRC = 0xff;
  DDRA = 0;

  //init the task timer
  time1=10;
  mode = mode_highsync;


  //crank up the ISRs
  sei();

end
//=====================================================
```