Improved Wireless Dog Fence

SD1522

Carl Baumann, Erik Brekkestran, and Cooper Willis

Contents

[Background 4](#_Toc450646033)

[ProjectRequirements 4](#_Toc450646034)

[Design Implementation Method 4](#_Toc450646035)

[Basic design idea: 4](#_Toc450646036)

[Post Placement: 5](#_Toc450646037)

[Corner Placement: 5](#_Toc450646038)

[Pictures of Device 6](#_Toc450646039)

[Schematics 7](#_Toc450646040)

[Master Post: 7](#_Toc450646041)

[Slave Posts: 8](#_Toc450646042)

[Collar/Handheld Unit: 9](#_Toc450646043)

[Software 10](#_Toc450646044)

[Master Post: 10](#_Toc450646045)

[Slave Posts: 10](#_Toc450646046)

[Collar/Handheld Unit: 10](#_Toc450646047)

[Technician’s Troubleshooting Section 11](#_Toc450646048)

[Project Comments 12](#_Toc450646049)

[Problems: 12](#_Toc450646050)

[Design Flaws: 12](#_Toc450646051)

[Future Improvements: 12](#_Toc450646052)

[Projects That Could Be Derived From Our Work: 12](#_Toc450646053)

[Tips for New Design Students: 12](#_Toc450646054)

[Appendix 13](#_Toc450646055)

[PCB Layout: 13](#_Toc450646056)

[Master Post: 13](#_Toc450646057)

[Slave Posts (Used Design): 13](#_Toc450646058)

[Slave Posts (Post Mortem Design): 14](#_Toc450646059)

[Collar/Handheld Unit (Used Design): 14](#_Toc450646060)

[Collar/Handheld Unit (Post Mortem Design): 15](#_Toc450646061)

[Software Code: 15](#_Toc450646062)

[Master Post: 15](#_Toc450646063)

[Slave Post 1: 17](#_Toc450646064)

[Slave Post 2: 20](#_Toc450646065)

[Collar/Handheld Unit: 22](#_Toc450646066)

[Parts List: 31](#_Toc450646067)

[Data Sheets: 31](#_Toc450646068)

# Background

This project’s main goal was to create an affordable wireless dog fence capable of creating and reinforcing a polygonal border. Most of the current market alternatives create a circular border, or a combination of multiple circular borders, that the dog is allowed to be in. There are polygonal based wireless dog fences that already exist, but these systems are either expensive (with a cost of around a thousand dollars) or require a wire to be buried where the border will be.

# ProjectRequirements

* Wireless (i.e. no buried wire)
* Usable by a non-technical consumer
* Utilizes a minimum of three transmitters/transceivers
* Definable polygonal border
* Collar and/or handheld unit with border violation indication
* Affordable Cost
* Range that can accommodate a typical urban yard size
* Battery powered transmitters/transceivers and collar/handheld unit

# Design Implementation Method

## F:\DesignProjectTestStuffImages\Techical_Document_Figure_Border_and_Post_Placement.pngBasic design idea:

The design method chosen was to use three stationary posts placed inside the border area, similar to those shown in **Figure 1**. These posts send out timed pulses which are received by the collar/handheld unit. The collar/handheld unit then reads the power received from each of the pulses to roughly determine its distance from each of the posts. It compares this distance against stored values to check if it is inside the border or outside the border.

Figure 1: Desired border (black line) and the actual border (pink area), after setting a border point at each of the corners

## Post Placement:

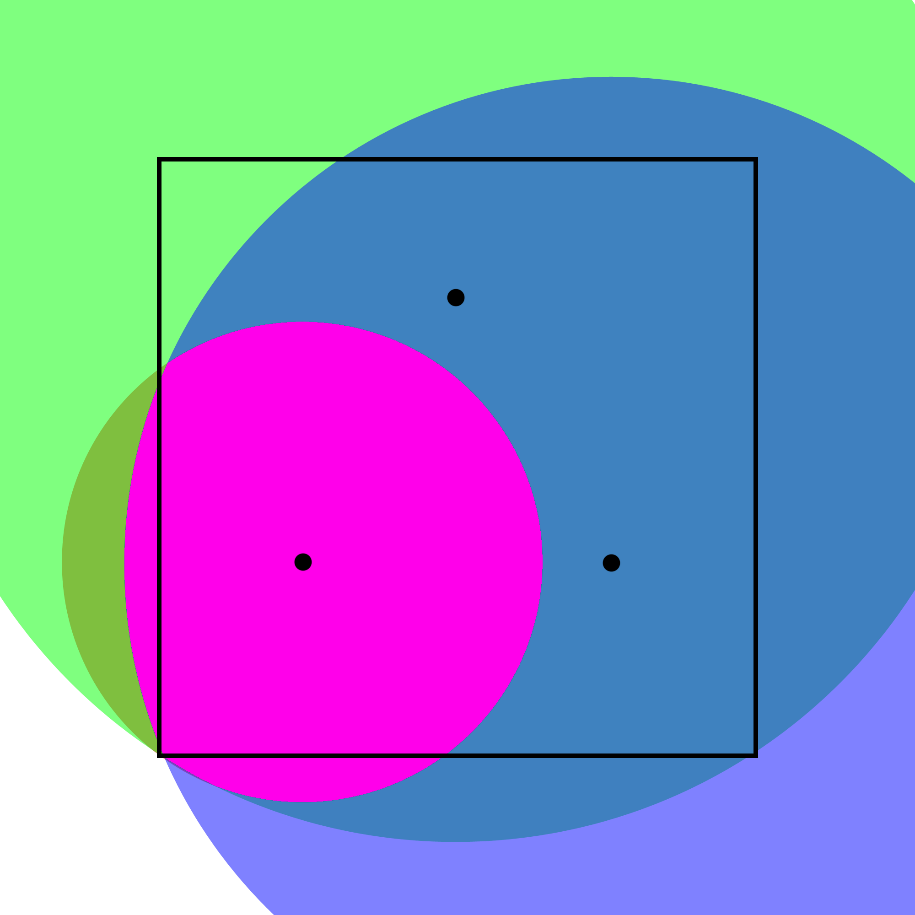
****The vertical spacing between the uppermost post and the upper most point of the border is roughly one third of the total height of the border. A similar distance is used for the vertical spacing between the bottom posts and the lowest point on the border. The horizontal spacing for the uppermost post should be roughly half way between the outermost points to its left and right. The horizontal spacing on the bottom points is similar to the vertical space where the leftmost post is placed a third of the way in from the leftmost point on the border. The rightmost post is placed in a similar fashion to leftmost one, being placed a third of the way from the rightmost point on the border. This placement method creates the best overlapping of coverage areas between the three posts.

Figure 2: Border area from a single corner point (pink) and the corresponding power radii for each post (in red, green, and blue overlays)

If the posts are placed too close to the outside of the border there will be “dead areas” (which act like they are not a part of the border) in the center of the border, and if the posts are all placed in the center of the area, then the actual border will become more circular. **Figure 2** shows an example of how the border is built based on the posts’ locations relative to the border points being set. **Figure 4** and **Figure 5** show the posts and their enclosures, which are push into the ground.

## Corner Placement:

After the posts have been placed, pressing RB0 (labeled by “Distances” on the enclosure seen in **Figure 3**) will put the collar/handheld unit into a “Build mode”. While in Build mode, pressing RB1 (labeled by “Border Points” on the enclosure) will set a new border point in the Collar/Handheld unit.

# Pictures of Device

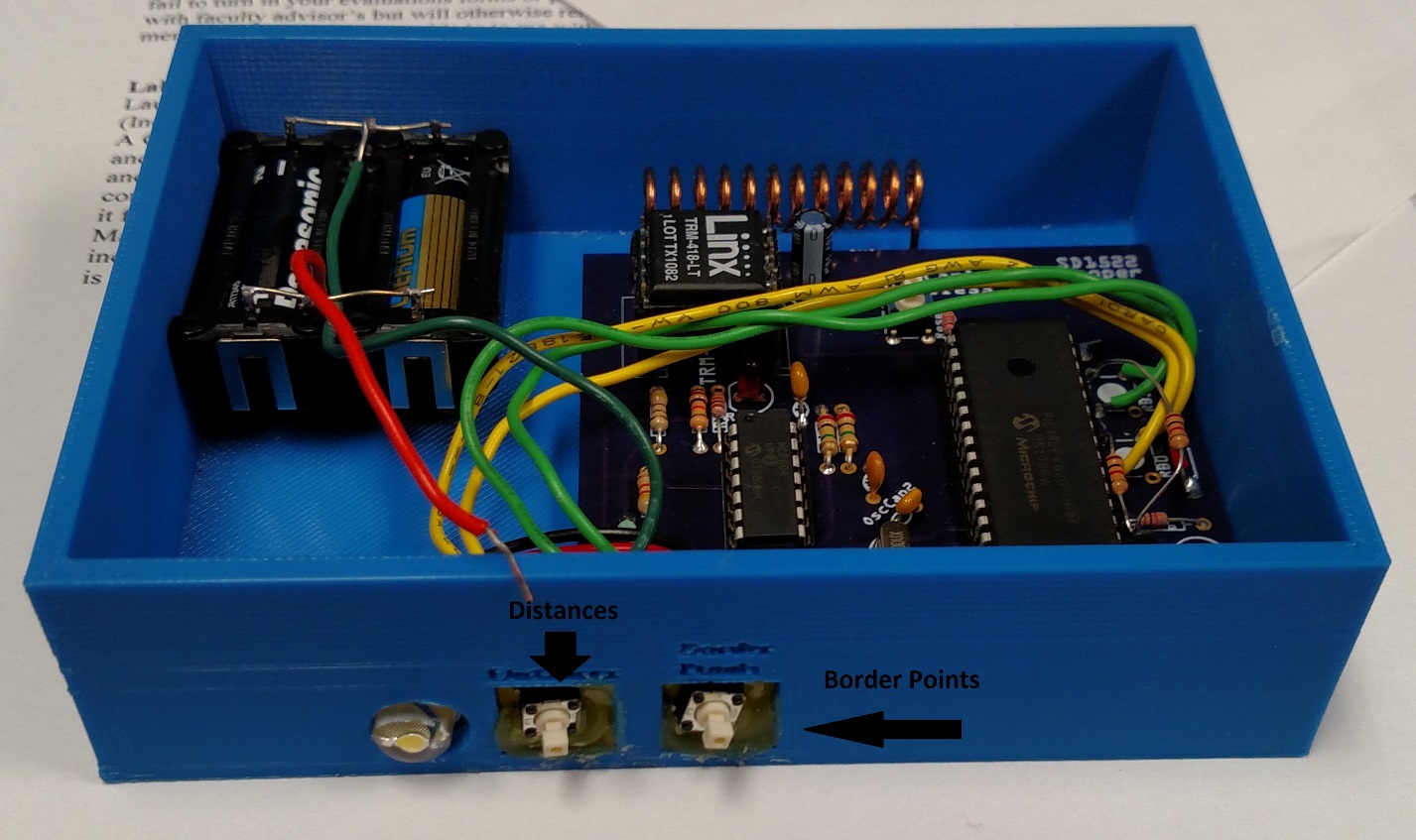


Figure 3: The Collar, in its handheld form, with the buttons and the output LED visible

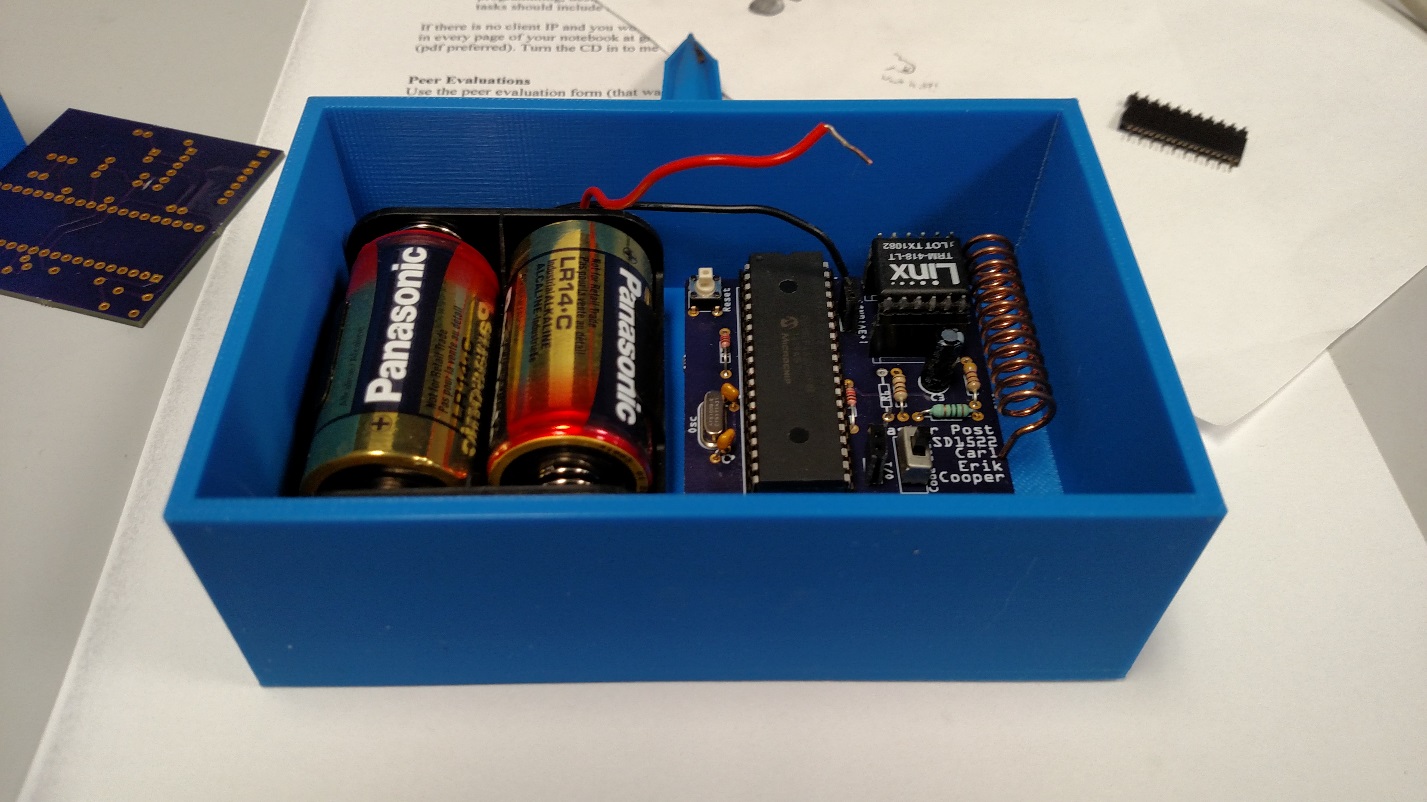


Figure 4: The Master Post with the stake visible on the top

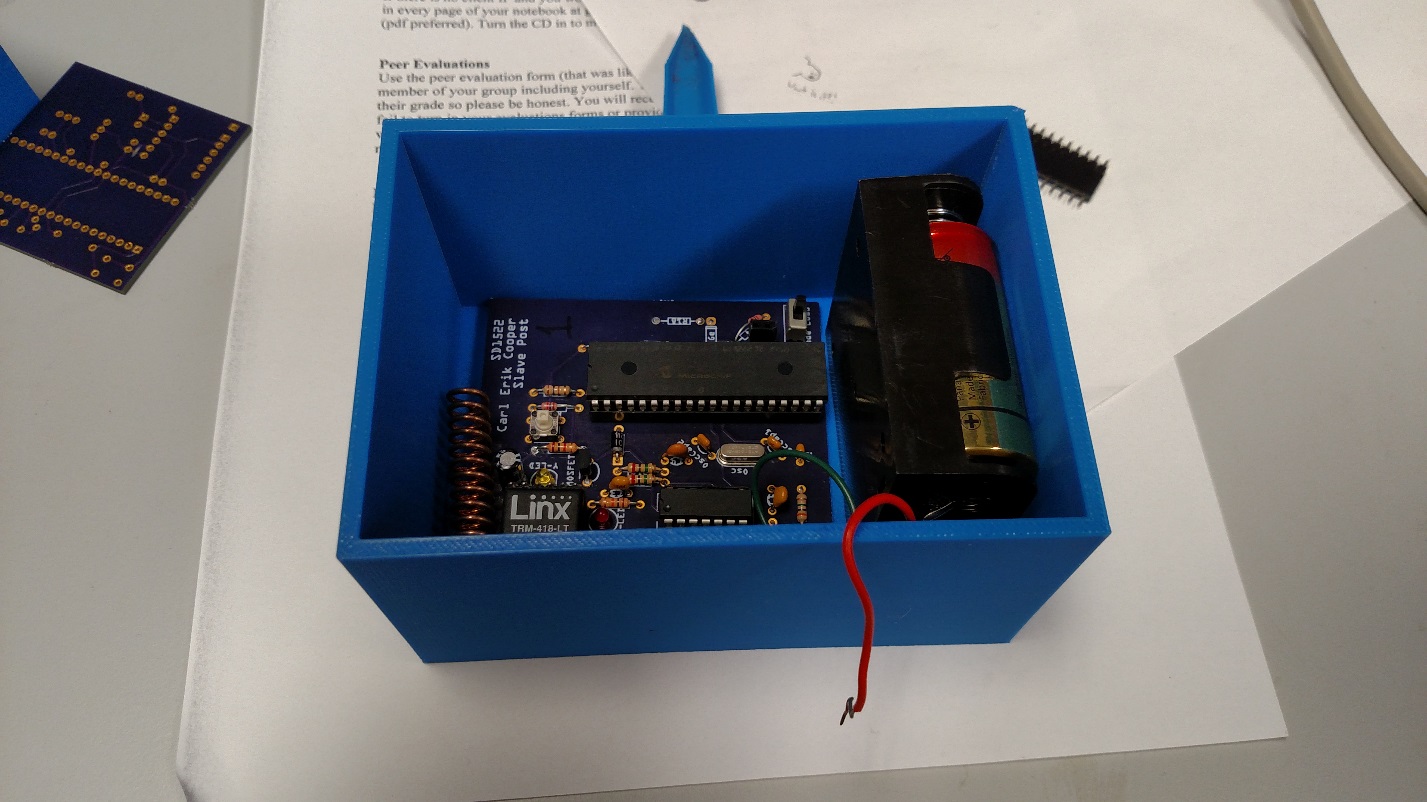


Figure 5: A Slave Post with the stake visible on the top

# Schematics

## Master Post:

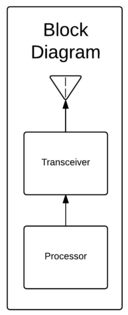
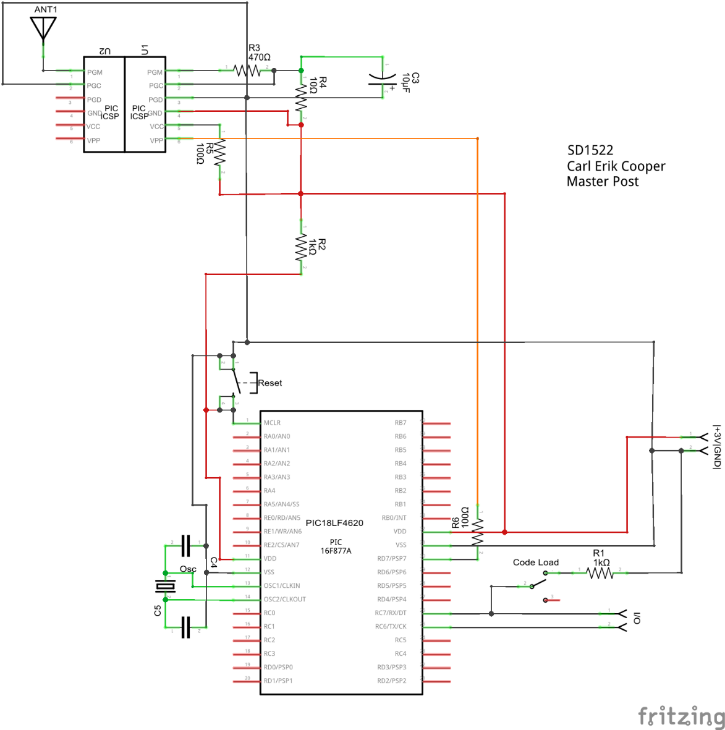
As can be seen in **Figure 6**, the Master Post has a very simple design. It is essentially just a timer connected to the transceiver, which sends out a pulse every two seconds. **Figure 7** shows the more in depth circuit schematic for the Master Post.

Figure 6: Block diagram for the Master Post

Figure 7: Circuit schematic for Master Post

## Slave Posts:

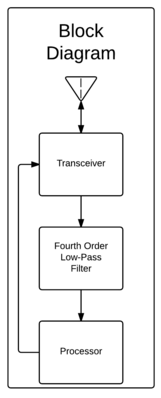
The Slave Posts, are a slightly more complicated version of the Master Post’s block diagram and circuit (as seen in **Figure 8** and **Figure 9**). The Slave Posts have to be able to send, as well as receive a signal. The signal being received has a lot of noise, so to remove the noise a fourth order low-pass filter was added.

Figure 8: Block diagram for Slave Posts

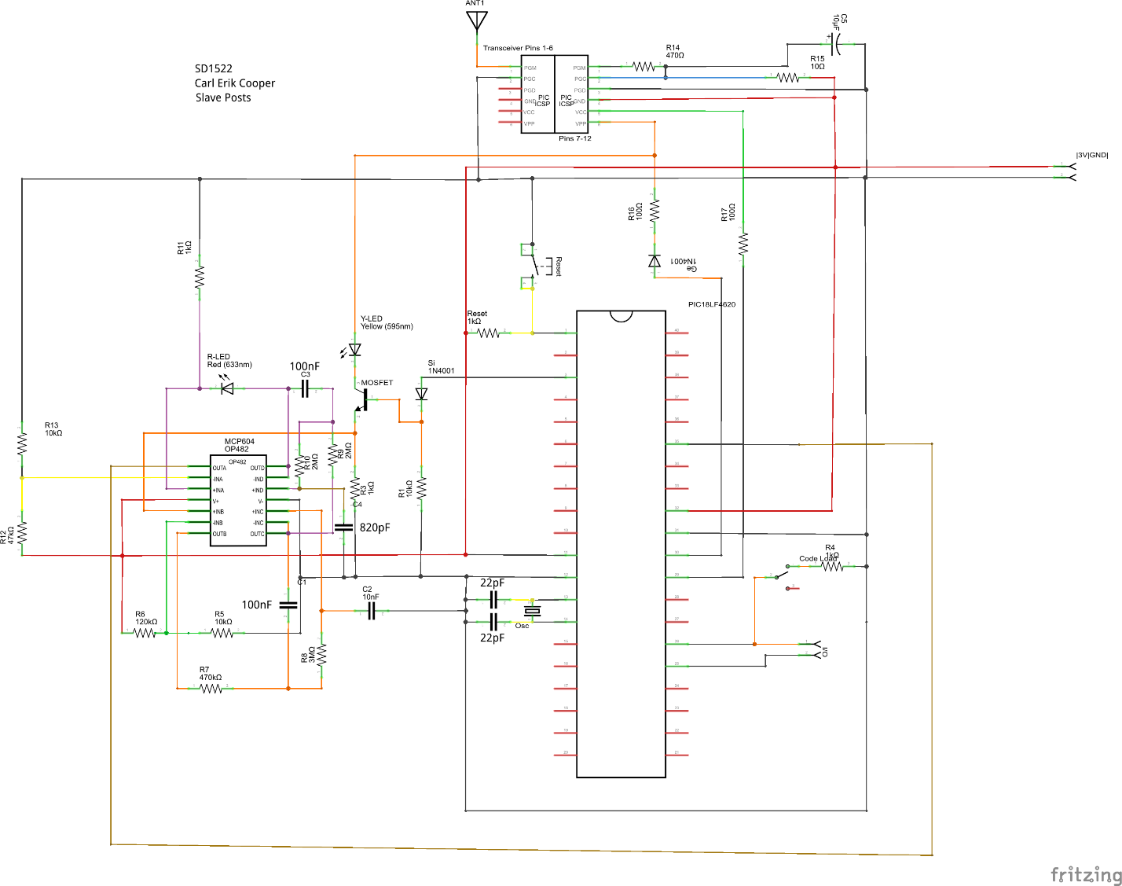
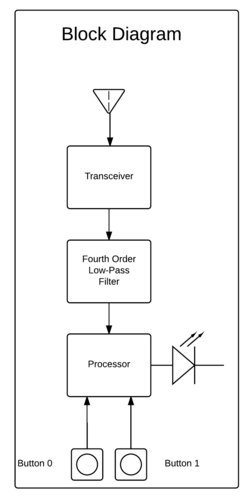


Figure 9: Circuit schematic for Slave Posts

## Collar/Handheld Unit:

The Collar/Handheld Unit, is a modification of the Slave Post circuit, as seen in **Figure 10** and **Figure 11**. There is an added LED, and two additional buttons, for user input. The fourth order low-pass filter is still required, since the Controller will be receiving.

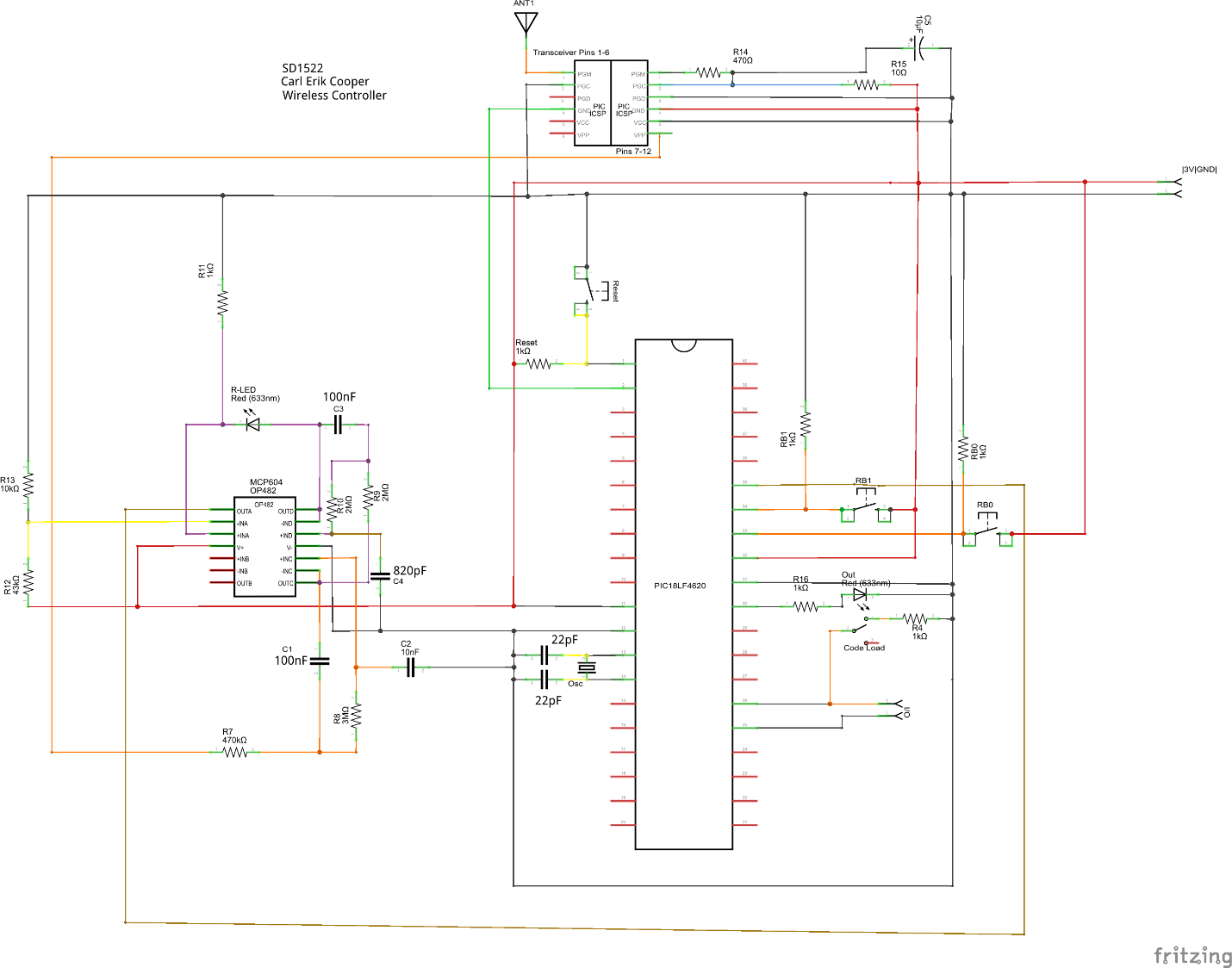
Figure 10: Block diagram for Collar/Handheld Unit

Figure 11: Circuit schematic for Collar/Handheld Unit

# Software

## Master Post:

The Master Post sets the timing for the post transmission. It does this using a counter and timer interrupt. This signal is picked up by the Slave Posts and the Collar/Handheld Unit.

## Slave Posts:

The slave posts wait until they receive a signal from the Master Post. When they receive the pulse from the Master Post, they wait a specified time (different times for each of the Slave Posts) and then transmit their own signal.

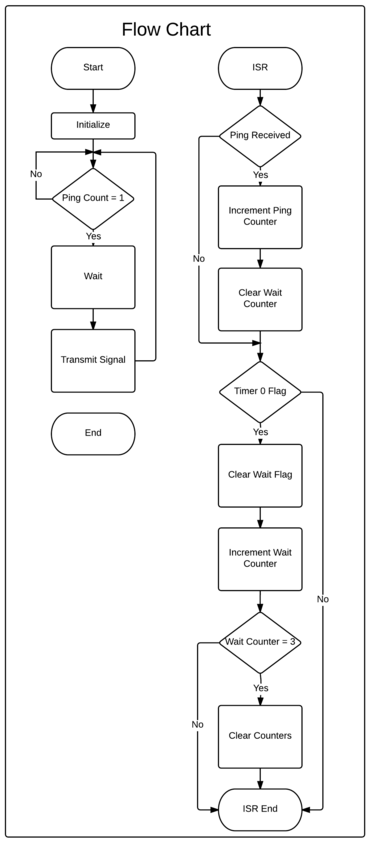


Figure 12: Flow chart for Master Post code

## Collar/Handheld Unit:

The Collar/Handheld Unit receives the signals sent out by the Posts and measures the signal strength to determine the distances to each of the posts.

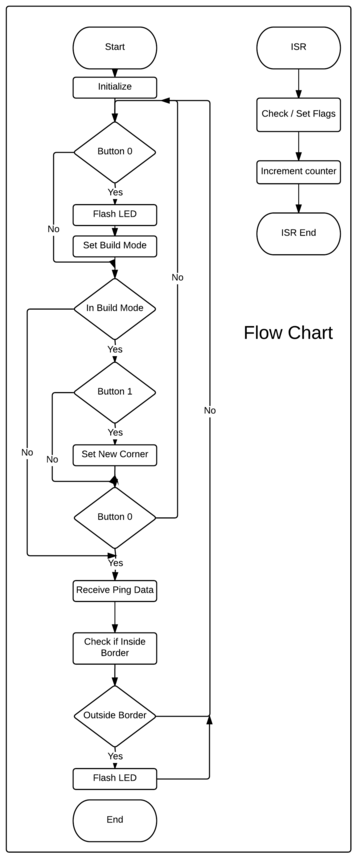
The border is also defined using the Collar/Handheld Unit. This is done by pressing the button labeled “Distances” on the side of the Collar/Handheld Unit (visible in **Figure 3)**, to enter the build mode for the border. The border is then defined by bringing the Collar/Handheld Unit to the location where the user would like to place a corner and pressing the button labeled “Border Points” (visible **Figure 3)**. This process is repeated until all of the desired corners have been recorded by the Collar/Handheld Unit (with a maximum of eight corners). Each time the Border Points button is pressed the Collar/Handheld Unit is recording the power being received by the transceiver from each of the posts. After setting all border points, pressing the Distances button again to exit build mode. The Collar/Handheld Unit then interprets this data to construct an area that lies within all of the border points. The recorded power received values are used to create three circular areas corresponding to each post for each corner placed, see **Figure 2**. Then the three circular areas for each corner are intersected together to form the smallest (pink) circle which is the relevant area for each corner, see **Figure 2**. All of the border points’ areas are then unioned together to create the total area that is defined to be within the border, see **Figure 1**.

Figure 13: Flow chart for Slave Post code

While the Collar/Handheld Unit isn’t in build mode it will continuously check the power levels of the received signal and will determine if those power levels correspond to a position inside the defined border or not. If there is no border defined the Collar/Handheld Unit will default to indicating that it is inside of the border.

# Technician’s Troubleshooting Section

Known fixable issues:

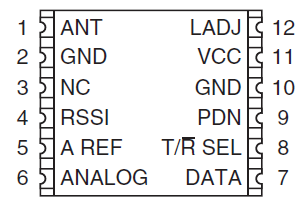
1. **Problem**: Collar/Handheld Unit doesn’t receive the signal from posts that are staked to the ground or on the floor of the first floor of a building.  
   **Solution**: Elevate the posts off of the ground/floor
2. **Problem**: Slave Post 1 or 2 is not receiving the signal transmitted by Master Post (i.e. the red light is not blinking every two seconds) ***OR*** Collar/Handheld Unit is not receive all three signals ***OR*** the Collar\Handheld Unit is receive a “bad” signal (i.e. the red light is not blinking three times every two seconds)  
   **Solution**: Check if 4 applies if not, short Transceiver Pin 11 to Transceiver Pin 10 on the Slave Post temporarily

Figure 14: Flow Chart for Collar/Handheld Unit

Figure 15: The pin output of the transceiver. On the physical package, pin 1 is indicated by the raised bump and the small “1”

# Project Comments

## Problems:

1. See Troubleshooting Section

## Design Flaws:

1. Power received indicator on transceiver being used for distance finding isn’t very accurate.

## Future Improvements:

1. Make the circuit small enough to be usable on a dog collar
2. Change Collar/Handheld Unit Code so that the borders of the actual area are flat
3. Add a voltage regulator to the power input
4. Change antennas from helical to whip for better range and signal clarity
5. Increase the transmission rate on the Posts so a more accurate position can be found
6. Change to surface mount parts
7. Removal of debugging parts (i.e. the code load switch, i/o header and red LED)
8. Add a pull down resister to filter output line
9. Change master post to a 555 timer or simpler microcontroller
10. Adding a Bluetooth component so that you could get location information on your phone
11. Powerless data storage
12. Power optimization

## Projects That Could Be Derived From Our Work:

A project that can be derived from our work would consist of implementing the above mentioned improvements.

## Tips for New Design Students:

1. Make sure rising edge interrupts are not used on pins that are left floating when not pulled high
2. If it’s working, take a video. It might not be working tomorrow.

# Appendix

## PCB Layout:

### Master Post:

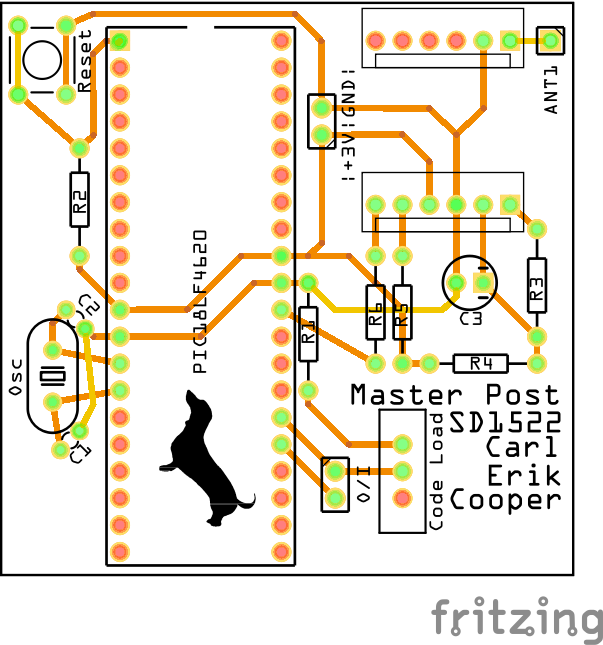


Figure 16: PCB layout for Master Post

### Slave Posts (Used Design):

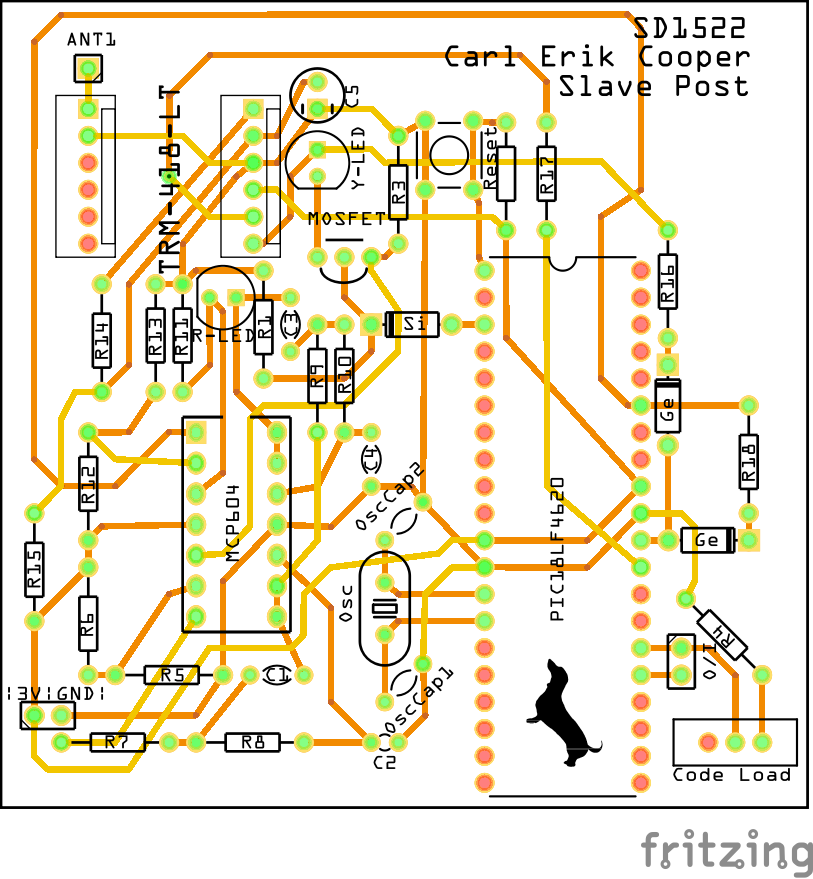


Figure 17: Slave Post PCB layout used for project. Note that R18 and the connected GE diode were not used in the final product

### Slave Posts (Post Mortem Design):

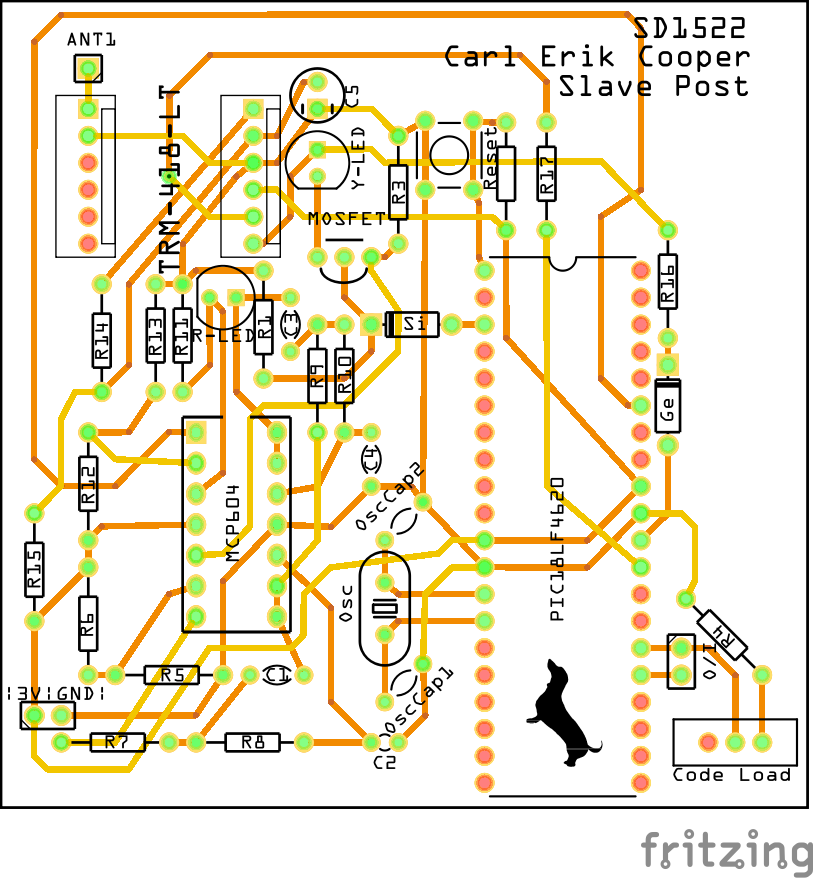


Figure 18: Revised Slave Post PCB Layout, revised after Demo Day

### Collar/Handheld Unit (Used Design):

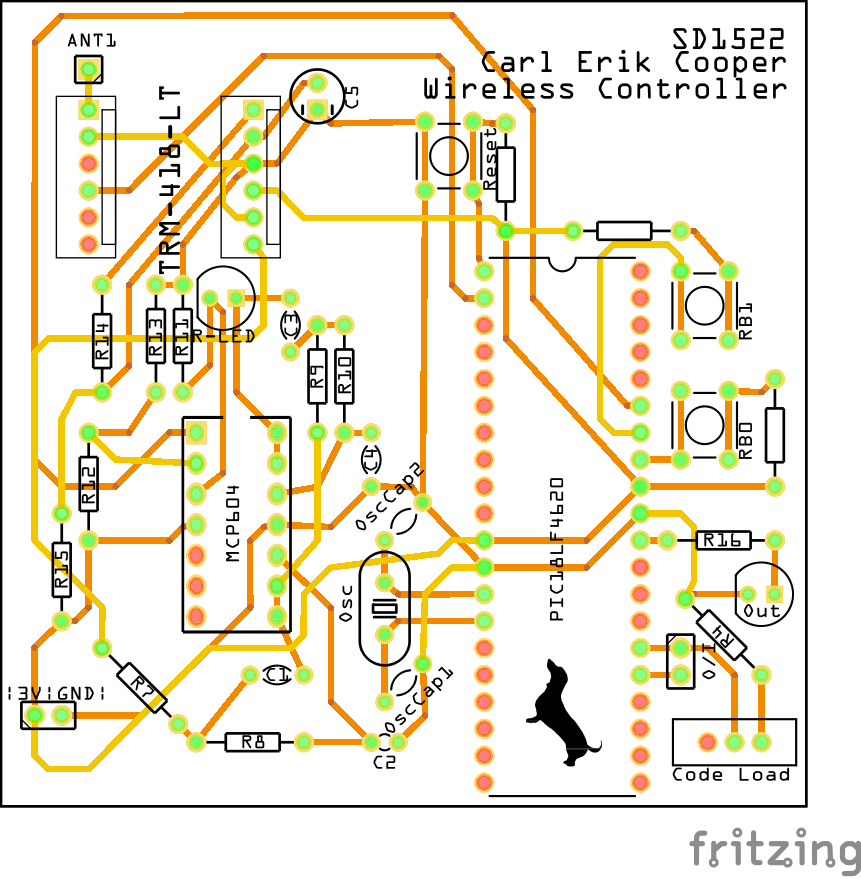


Figure 19: Collar/Handheld Unit PCB layout used for project

### Collar/Handheld Unit (Post Mortem Design):

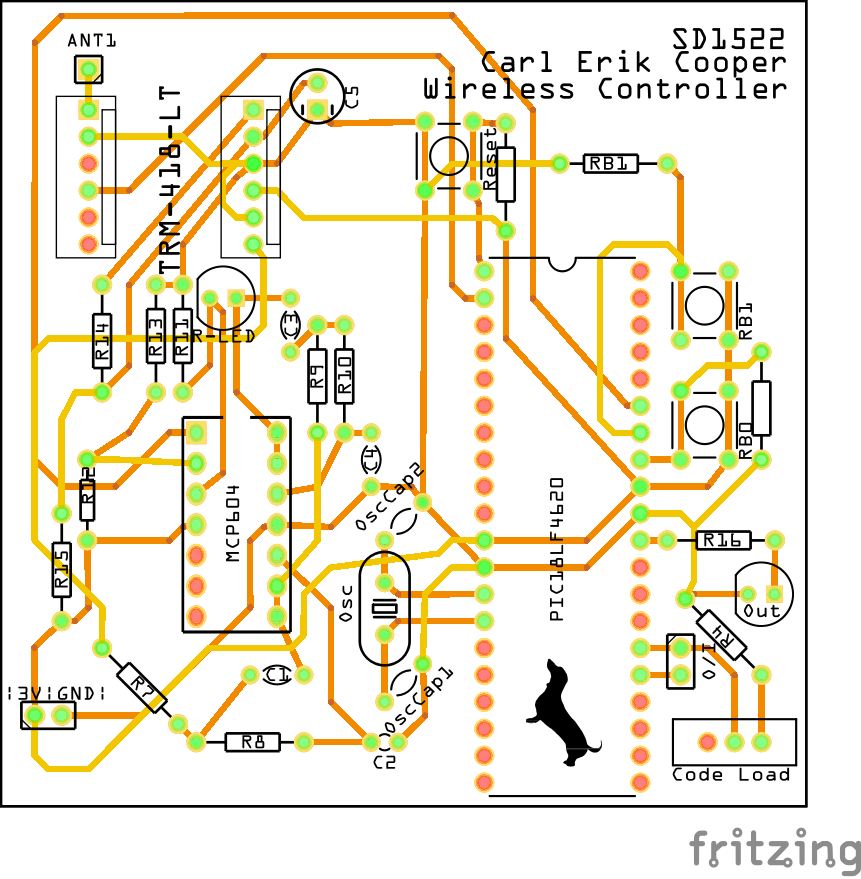


Figure 20: Revised Collar/Handheld Unit PCB layout, revised after Demo Day

## Software Code:

### Master Post:

//Tripost-master\_post

//Ping at constant timed rate

#include<pic18.h>

//globals

char InterruptCounter = 0;

//declare subroutines

void InitializePorts(void);

void InitializeISR(void);

void InitializeTransceiver(void);

void SetTransceiverSignal(void);

void ClearTransceiverSignal(void);

void IncrementInterruptCounter(void);

void ClearTMR0Flags(void);

void SetToTransmit(void);

//isr

void interrupt IntServe(void){

    if(TMR0IF){

        if(InterruptCounter == 0){

            SetTransceiverSignal();

        }

        else if(InterruptCounter == 1){

            ClearTransceiverSignal();

        }

        IncrementInterruptCounter();

        ClearTMR0Flags();

    }

}

//main

void main(void){

    InitializePorts();

    InitializeISR();

    InitializeTransceiver();

    while(1){

        //Do nothing - Wait for interrupts

    }

}

//subroutines

void InitializePorts(void){

    //Set input/output

    //0 is output, 255 is input

    TRISA = 0;

    TRISB = 0;

    TRISC = 0;

    TRISD = 0;

    TRISE = 0;

    //Clear ports

    PORTA = 0;

    PORTB = 0;

    PORTC = 0;

    PORTD = 0;

    PORTE = 0;

    //Set ports to digital

    ADCON0 = 0x00; // Analog disabled

    ADCON1 = 0x0F; // All input binary

    ADCON2 = 0x87; // ADC Clock Settings (not used)

}

void InitializeISR(void){

    //Enable bits for timer 0

    TMR0ON = 1;//timer 0 enable 1/3

    TMR0IE = 1;//timer 0 enable 2/3

    TMR0IP = 1;//timer 0 enable 3/3

    PEIE = 1;//timer interrupt enable

    GIE = 1;//global interrupt enable

    T0CS = 0; //Timer Oscillator Select, run off crystal

    T0CON = 0x85; //timer 0 prescaler, PS = 64

    TMR0 = -4883; //Set to 1/8th of a second for prescaler 64

}

void InitializeTransceiver(void){

    SetToTransmit();

}

void SetTransceiverSignal(void){

    RD7 = 1; //Turn on tranceiver's signal

}

void ClearTransceiverSignal(void){

    RD7 = 0; //Turn off tranceiver's signal

}

void IncrementInterruptCounter(void){

    InterruptCounter =  (InterruptCounter+1) & 0x0f; //Increment, reset at 16

}

void ClearTMR0Flags(void){

    TMR0IF = 0; //Clear timer flag

    TMR0 = -4883; //reset timer

}

void SetToTransmit(void){

    RD6 = 1; //Set tranceiver to transmit mode

}

### Slave Post 1:

// Tripost-Slave\_one

// Ping after master

#include<pic18.h>

// globals

char PingCounter = 0;

char WaitResetCounter = 0;

char TMR0WaitFlag = 1;

char i = 0;

// constants

const char SLAVE\_NUMBER = 1;

// declare subroutine

void InitializePorts(void);

void InitializeInterrupts(void);

void InitializeTransceiver(void);

void SetToReceive(void);

void SetToTransmit(void);

void ClearINT2Flags(void);

void ClearTMR0Flags(void);

void TransmitSignal(void);

void TMR0CountdownReset(void);

// isr

void interrupt IntServe(void){

    if(INT2IF)

    {

        PingCounter++;

        WaitResetCounter = 0;

        ClearINT2Flags();

    }

    if(TMR0IF){

        TMR0WaitFlag = 0;

        WaitResetCounter++;

        if(WaitResetCounter >= 5){

            WaitResetCounter = 0;

            PingCounter = 0;

        }

        ClearTMR0Flags();

    }

}

//main

void main(void){

    InitializePorts();

    InitializeInterrupts();

    InitializeTransceiver();

    while(1){ // main loop

        if(PingCounter == 1) { RD2 = 1; }

        else {RD2 = 0; }

        if(PingCounter == 2) { RD3 = 1; }

        else { RD3 = 0; }

        if(PingCounter == SLAVE\_NUMBER){

            SetToTransmit();

            // wait for rest of input signal

            TMR0CountdownReset();

            TMR0WaitFlag = 1;

            while(TMR0WaitFlag);

            // wait extra 1/8th of second to send

            TMR0WaitFlag = 1;

            while(TMR0WaitFlag);

            TransmitSignal();

            SetToReceive();

            for(i = 0;i<4;i++){

                TMR0WaitFlag = 1;

                while(TMR0WaitFlag);

            }

        }

    }

}

//subroutines

void InitializePorts(void){

    //Set input/output

    //0 is output 255 is input

    TRISA = 0;

    TRISB = 255;

    TRISC = 0;

    TRISD = 0;

    TRISE = 0;

    //Clear ports

    PORTA = 0;

    PORTB = 0;

    PORTC = 0;

    PORTD = 0;

    PORTE = 0;

    //Set ports to digital

    ADCON0 = 0x00; // Analog disabled

    ADCON1 = 0x0F; // All input binary

    ADCON2 = 0x87; // ADC Clock Settings (not used)

}

void InitializeInterrupts(void){

    GIE = 1;//Global interupt enable

    PEIE = 1;//timer interrupt enable

    //turn on rising edge interrupt on RB2 (INT2)

    INTEDG2 = 1;//Set to rising edge interrupt

    INT2IE = 1;//int2 enable 1/3

    INT2IP = 1;//int2 enable 2/3

    TRISB2 = 1;//int2 enable 3/3

    //turn on timer 0

    TMR0ON = 1;//timer 0 enable 1/3

    TMR0IP = 1;//timer 0 enable 2/3

    TMR0IE = 1;//timer 0 enable 3/3

    T0CS = 0;//use osc/4

    T0CON = 0x85;//timer 0 prescaler, set to 64

    TMR0 = -4883;//set to 1/8 sec for prescaler 64

}

void SetToReceive(void){

    RD6 = 0;//set transceiver to receive mode

    RA1 = 1;//turn on switch on line c

}

void SetToTransmit(void){

    RA1 = 0;//turn off switch on line c

    RD6 = 1;//set transceiver to transmit mode

}

void InitializeTransceiver(void){

    SetToReceive();

}

void ClearINT2Flags(void){

    INT2IF = 0;

}

void ClearTMR0Flags(void){

    TMR0IF = 0;

    TMR0 = -4883;

}

void TMR0CountdownReset(void){

    TMR0 = -4883;

}

void TransmitSignal(void){

    RD7 = 1;

    TMR0CountdownReset();

    TMR0WaitFlag = 1;

    while(TMR0WaitFlag);

    RD7 = 0;

}

### Slave Post 2:

// Tripost-Slave\_two

// Ping after master and slave post one

//post turn on sequence

    //Both Slaves

    //Master

#include<pic18.h>

// globals

char PingCounter = 0;

char WaitResetCounter = 0;

char TMR0WaitFlag = 1;

// declare subroutine

void InitializePorts(void);

void InitializeInterrupts(void);

void InitializeTransceiver(void);

void SetToReceive(void);

void SetToTransmit(void);

void ClearINT2Flags(void);

void ClearTMR0Flags(void);

void TransmitSignal(void);

void TMR0CountdownReset(void);

// isr

void interrupt IntServe(void){

    if(INT2IF)

    {

        PingCounter++;

        WaitResetCounter = 0;

        ClearINT2Flags();

    }

    if(TMR0IF){

        TMR0WaitFlag = 0;

        WaitResetCounter++;

        if(WaitResetCounter == 5){

            WaitResetCounter = 0;

            PingCounter = 0;

        }

        ClearTMR0Flags();

    }

}

//main

void main(void){

    InitializePorts();

    InitializeInterrupts();

    InitializeTransceiver();

    while(1){ // main loop

        if(PingCounter == 1){

            SetToTransmit();

            // wait for rest of input signal from master

            TMR0CountdownReset();

            TMR0WaitFlag = 1;

            while(TMR0WaitFlag);

            //hard code wait

            // wait extra 3/8th of second to send

            TMR0WaitFlag = 1;

            while(TMR0WaitFlag);

            TMR0WaitFlag = 1;

            while(TMR0WaitFlag);

            TMR0WaitFlag = 1;

            while(TMR0WaitFlag);

            TransmitSignal();

            SetToReceive();

        }

    }

}

//subroutines

void InitializePorts(void){

    //Set input/output

    //0 is output 255 is input

    TRISA = 0;

    TRISB = 255;

    TRISC = 0;

    TRISD = 0;

    TRISE = 0;

    //Clear ports

    PORTA = 0;

    PORTB = 0;

    PORTC = 0;

    PORTD = 0;

    PORTE = 0;

    //Set ports to digital

    ADCON0 = 0x00; // Analog disabled

    ADCON1 = 0x0F; // All input binary

    ADCON2 = 0x87; // ADC Clock Settings (not used)

}

void InitializeInterrupts(void){

    GIE = 1;//Global interupt enable

    PEIE = 1;//timer interrupt enable

    //turn on rising edge interrupt on RB2 (INT2)

    INTEDG2 = 1;//Set to rising edge interrupt

    INT2IE = 1;//int2 enable 1/3

    INT2IP = 1;//int2 enable 2/3

    TRISB2 = 1;//int2 enable 3/3

    //turn on timer 0

    TMR0ON = 1;//timer 0 enable 1/3

    TMR0IP = 1;//timer 0 enable 2/3

    TMR0IE = 1;//timer 0 enable 3/3

    T0CS = 0;//use osc/4

    T0CON = 0x85;//timer 0 prescaler, set to 64

    TMR0 = -4883;//set to 1/8 sec for prescaler 64

}

void SetToReceive(void){

    RD6 = 0;//set transceiver to receive mode

    RA1 = 1;//turn on switch on line c

}

void SetToTransmit(void){

    RA1 = 0;//turn off switch on line c

    RD6 = 1;//set transceiver to transmit mode

}

void InitializeTransceiver(void){

    SetToReceive();

}

void ClearINT2Flags(void){

    INT2IF = 0;

}

void ClearTMR0Flags(void){

    TMR0IF = 0;

    TMR0 = -4883;

}

void TMR0CountdownReset(void){

    TMR0 = -4883;

}

void TransmitSignal(void){

    RD7 = 1;

    TMR0CountdownReset();

    TMR0WaitFlag = 1;

    while(TMR0WaitFlag);

    RD7 = 0;

}

### Collar/Handheld Unit:

|  |
| --- |
| Utilities.h |

1 /\*

2 \* File: Utilities.h

3 \* Author: carlbaumann

4 \*

5 \* Created on April 22, 2016, 8:25 AM

6 \*/

7

8 #ifndef UTILITIES\_H

9 #define UTILITIES\_H

10

11 /\*\*\*\*\*\*\*\*\*\*Utility Types/Constants\*\*\*\*\*\*\*\*\*\*\*/

12

13 #define false 0

14 #define true 1

15

16 typedef struct {

17 int masterValue;

18 int slave1Value;

19 int slave2Value;

20 } triPingData;

21

22 typedef int bool;

23

24

25 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*Utility Functions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

26 void OutputToSerial(int data);

27 int AbsoluteValue\_int(int num);

28 unsigned int A2D\_Read(void);

29

30

31 void OutputToSerial(int data){

32 unsigned char A[5];

33 if(data < 0) { //output '-' if the value is negative

34 while(!TRMT);

35 TXREG = 45;

36 }

37 for (int i=0; i<5; i++) { //get 4 decimal places stored into array

38 A[i] = (unsigned char)AbsoluteValue\_int(data) % 10;

39 data = AbsoluteValue\_int(data) / 10;

40 }

41 for (int i=5; i>0; i--) {

42 while(!TRMT);

43 TXREG = A[i-1] + 48; //Output ASCII number

44 }

45

46 while(!TRMT);

47 TXREG = ' ';

48 return;

49 }

50

51 int AbsoluteValue\_int(int num){

52 if (num < 0){

53 return -num;

54 }

55 else{

56 return num;

57 }

58 }

59

60 // reads and returns the value from the A2D converter

61 unsigned int A2D\_Read(void) {

62 ADCON0 = 0x01; //select channel 0 for a2d reading

63 GODONE = 1; //start conversions

64 while (GODONE); //wait until done (approx 8us)

65 return (ADRES);

66 }

67

68

69 #endif /\* UTILITIES\_H \*/

70

71

72

|  |
| --- |
| collar\_lookup\_array.c |

1 /\* Use instructions:

2 \* After reseting the controller, press RB0 to enter Build Mode

3 \* While in Build Mode, the LED will flicker

4 \* After pressing RB1 the controller will collect data and set corner point

5 \* After the corner point was succefully set the LED will flash and start

6 \* to flicker again

7 \* Once all 4 corner points have been set, the LED will flash multiple times

8 \* Pressing RB0 while in Build mode will exit build mode

9 \* If the controller is not in build mode, it will constantly check borders

10 \* If the LED flashes, it is outside the border.

11 \* (the flash simulates a shock being administered)

12 \*

13 \* Inputs - RB0 and RB1

14 \* Outputs - RD7 (green LED)

15 \*/

16

17 #include<pic18.h>

18 #include"Utilities.h"

19

20

21 unsigned int A2D\_Read(void);

22 void InitializeInterrupts(void);

23 void InitializePorts(void);

24 void ResetTMR0Countdown(void);

25 void WaitForPingBreak(void);

26 void OutputFourCornerPointsToSerial(void);

27 void FlashLED(int numberOfTimes);

28 void SetNewPoint(triPingData newPoint);

29

30 bool CheckIfInsideCornerBorder(int cornerNumber, triPingData pingData);

31 bool CheckIfInsideBorders(triPingData pingData);

32

33 triPingData RetrieveNextPingsData();

34

35 bool didReceivePing = false;

36 bool isPingBreak = false;

37 bool isInBuildMode = false;

38 bool areAllCornersSet = false;

39

40 bool wasButton0Pressed = false;

41 bool wasButton1Pressed = false;

42

43 char pingCounter = 0;

44 char array\_index = 0;

45

46 int lightCounter = 0;

47 int numberOfCorners = 0;

48 //int maxNumberOfCorners = 8;

49

50 triPingData cornerPoints[8];

51 triPingData mostRecentPingData;

52

53 // all interrupts are handled here

54 void interrupt IntServe(void) {

55 if(INT0IF){

56 // debouncing test - see if noise or actual button press

57 for(int i = 0; i < 32000; i++);

58 if(RB0){

59 wasButton0Pressed = true;

60 }

61 else { wasButton0Pressed = false; }

62

63 INT0IF = 0;

64 }

65 if(INT1IF){

66 // debouncing test - see if noise or actual button press

67 for(int i = 0; i < 32000; i++);

68 if(RB1){

69 wasButton1Pressed = true;

70 }

71 else { wasButton1Pressed = false; }

72

73 INT1IF = 0;

74 }

75 if(INT2IF){

76 didReceivePing = true;

77 pingCounter++;

78

79 if(pingCounter == 3) {

80 ResetTMR0Countdown();

81 isPingBreak = true;

82 }

83 else if(pingCounter > 3 ) {

84 pingCounter = 0;

85 }

86

87 // clear interrupt flag

88 INT2IF = 0;

89 }

90

91 if(TMR0IF){

92 // reset isPingBreak after 1 second has passed from ping counter == 3

93 if(isPingBreak){

94 pingCounter = 0;

95 isPingBreak = false;

96 }

97

98

99 //clear interrupt flag and reset to 1 second

100 TMR0IF = 0;

101 ResetTMR0Countdown();

102 }

103 }

104

105 void main(void) {

106 InitializeInterrupts();

107 InitializePorts();

108

109 // main loop

110 while(1){

111 /\* if RB0

112 \* wait for button

113 \* set flag for build

114 \* if build\_flag

115 \* turn light on

116 \* if RB1

117 \* turn light off

118 \* wait for button

119 \* RetrieveNextPingsData()

120 \* SetNewPoint()

121 \* if RB0

122 \* wait for RB0

123 \* clear flag for build

124 \* RetrieveNextPingsData()

125 \* CheckIfInsideBorders()

126 \*/

127

128 if(wasButton0Pressed) {

129 wasButton0Pressed = false;

130 isInBuildMode = true;

131 }

132 if(isInBuildMode) {

133 //flicker light

134 if(lightCounter++ > 1000){ RD7 = !RD7; lightCounter = 0; }

135

136 if(wasButton1Pressed) {

137 //turn light off

138 wasButton1Pressed = false;

139 RD7 = 0;

140 mostRecentPingData = RetrieveNextPingsData();

141 SetNewPoint(mostRecentPingData);

142 // FlashLED(numberOfCorners);

143 }

144

145 if(wasButton0Pressed) {

146 wasButton0Pressed = false;

147 isInBuildMode = false;

148 RD7 = 0;

149 }

150 }

151 else {

152 mostRecentPingData = RetrieveNextPingsData();

153

154 if(!CheckIfInsideBorders(mostRecentPingData)){

155 RD7 = 1;

156 for(int i = 0; i < 32000; i++);

157 for(int i = 0; i < 32000; i++);

158 for(int i = 0; i < 32000; i++);

159 RD7 = 0;

160 }

161 }

162 }

163 return;

164 }

165

166 // initialize all PIC variables for the interrupts

167 void InitializeInterrupts(void) {

168 GIE = 1; //Global interrupt enable

169 PEIE = 1; //Timer interrupt enable

170

171 //turn on rising edge interrupt on RB0 (INT0)

172 INTEDG0 = 1; //Set to rising edge interrupt

173 INT0IE = 1; //int0 enable 1/2

174 TRISB0 = 1; //int0 enable 2/2

175

176 //turn on rising edge interrupt on RB1 (INT1)

177 INTEDG1 = 1; //Set to rising edge interrupt

178 INT1IE = 1; //int1 enable 1/3

179 INT1IP = 1; //int1 enable 2/3

180 TRISB1 = 1; //int1 enable 3/3

181

182 //turn on rising edge interrupt on RB2 (INT2)

183 INTEDG2 = 1; //Set to rising edge interrupt

184 INT2IE = 1; //int2 enable 1/3

185 INT2IP = 1; //int2 enable 2/3

186 TRISB2 = 1; //int2 enable 3/3

187

188 //turn on Timer 0 interrupt

189 T0CS = 0; //timer 0 clock, use osc/4

190 T0CON = 0x87; //Timer 0 prescaler, PS = 256

191 TMR0 = -9766; //set to 1 sec for PS = 256

192 TMR0ON = 1; //Timer 0 enable 1/3

193 TMR0IE = 1; //Timer 0 enable 2/3

194 TMR0IP = 1; //Timer 0 enable 3/3

195

196 //turn on uart interrupt

197 //set baud rate to 9600 for 10MHz osc

198 TXIE = 0;

199 RCIE = 0;

200 BRGH = 0;

201 BRG16 = 1;

202 SYNC = 0;

203 SPBRG = 64;

204 TXSTA = 0x22;

205 RCSTA = 0x90;

206

207 return;

208 }

209

210 void InitializePorts(void) {

211 //set input/output

212 //0x00 is output, 0xff is input

213 TRISA = 0xff;

214 TRISB = 0xff;

215 TRISC = 0xc0; //set RC6 and RC7 to input for uart

216 TRISD = 0x00;

217 TRISE = 0x0f;

218

219 //clear ports

220 PORTA = 0;

221 PORTB = 0;

222 PORTC = 0;

223 PORTD = 0;

224 PORTE = 0;

225

226 //Set Ports B,C,D,E to digital, Set Port A to analog

227 ADCON0 = 0x01; // Analog on channel 0, RA0

228 ADCON1 = 0x07; // Analog/Binary Select

229 ADCON2 = 0x85; // ADC Clock Settings

230

231 return;

232 }

233

234

235 void ResetTMR0Countdown(void){

236 TMR0 = -9766;

237 return;

238 }

239

240 // sets the next corner point

241 void SetNewPoint(triPingData newPoint){

242 cornerPoints[numberOfCorners] = newPoint;

243

244 numberOfCorners++;

245

246 //cap the number of corners to 8

247 if(numberOfCorners > (8-1)){ numberOfCorners = 8; }

248

249 FlashLED(numberOfCorners);

250

251 return;

252 }

253

254 //checks the three parameters against the stored data, returns if inside

255 bool CheckIfInsideCornerBorder(int cornerNumber, triPingData pingData){

256 bool isInsideMasterPostRange = false;

257 bool isInsideSlave1PostRange = false;

258 bool isInsideSlave2PostRange = false;

259

260 if(pingData.masterValue > cornerPoints[cornerNumber].masterValue) { isInsideMasterPostRange = true; }

261

262 if(pingData.slave1Value > cornerPoints[cornerNumber].slave1Value) { isInsideSlave1PostRange = true; }

263

264 if(pingData.slave2Value > cornerPoints[cornerNumber].slave2Value) { isInsideSlave2PostRange = true; }

265

266 return (isInsideMasterPostRange && isInsideSlave1PostRange && isInsideSlave2PostRange);

267 }

268

269 // checks each corner point's data set to see if the dog is inside the bounds

270 bool CheckIfInsideBorders(triPingData pingData){

271 // OutputFourCornerPointsToSerial();

272 // if no corners are set, there is no border

273 if (numberOfCorners == 0) { return true; }

274

275 bool insideBorder = false;

276 for (int cornerNumber = 0; cornerNumber < numberOfCorners; cornerNumber++){

277 if (CheckIfInsideCornerBorder(cornerNumber, pingData)) {

278 insideBorder = true;

279 }

280 }

281 return insideBorder;

282

283 /\* retrieve the 3 pings of data from the posts store in variables

284 \* if any of the 4 points's data sets is inside

285 \* then the dog is inside, clear a flag and break from loop

286 \* if the dog isn't in any of the areas

287 \* then the dog is outside all of them, set flag

288 \* if the flag is set then turn on LED

289 \*/

290 }

291

292 // retrieves the ping data

293 triPingData RetrieveNextPingsData() {

294 triPingData readValues;

295 WaitForPingBreak();

296 while(pingCounter <= 3){

297 if(didReceivePing){

298 didReceivePing = false;

299 if(pingCounter == 1) {

300 readValues.masterValue = A2D\_Read();

301 }

302 else if(pingCounter == 2) {

303 readValues.slave1Value = A2D\_Read();

304 }

305 else if(pingCounter == 3) {

306 readValues.slave2Value = A2D\_Read();

307 break;

308 }

309 }

310 }

311 return readValues;

312 }

313

314 // outputs all corner points to serial (for debugging)

315 void OutputAllCornerPointsToSerial(){

316 for(int i = 0; i < numberOfCorners; i++) {

317 OutputToSerial(cornerPoints[i].masterValue);

318 OutputToSerial(cornerPoints[i].slave1Value);

319 OutputToSerial(cornerPoints[i].slave2Value);

320

321 while(!TRMT);

322 TXREG = 10; // line feed

323 while(!TRMT);

324 TXREG = 13; // carriage return

325 }

326 return;

327 }

328

329 // waits for the ping break to start and stop (used before collecting values)

330 void WaitForPingBreak(){

331 while(!isPingBreak); // wait for ping break flag

332 while(isPingBreak); // wait for ping break flag to clear

333 return;

334 }

335

336 /\* Generic Point Creation - (max of 8 points)

337 \* keep track of the number of points that are entered

338 \* when new point is added, store data into (numberOfPoints++)

339 \* (use variable++ to increment after use)

340 \* when checking, use for loop until at numberOfPoints

341 \* if numberOfPoints is not set (i.e. 0), then do nothing

342 \*

343 \* when setting a new point

344 \* flash LED once for each point stored

345 \*/

346

347 // flashes the output LED for numberOfTimes

348 void FlashLED(int numberOfTimes){

349 for(int i=0; i < numberOfTimes; i++){

350 RD7 = 0;

351 for(int i = 0; i < 32000; i++){ RD7 = 1; }

352 for(int i = 0; i < 32000; i++){ RD7 = 0; }

353 }

354 return;

355 }

356

## Parts List:

Table 1: Parts list with pricing from Digikey

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Manufacture Part Number | Quantity | Price per Unit (Retail) | Total Parts Price (Retail) | Price per unit (acquired) | Total Parts Price (Acquired) |
| MCP604-I/P | 3 | 1.04 | 3.12 | 0 | 0 |
| ZVN2106A | 2 | 0.68 | 1.36 | 0 | 0 |
| ANT-418-HETH | 4 | 1.12 | 4.48 | 1.12 | 4.48 |
| TRM-418-LT | 4 | 16.65 | 66.6 | 16.65 | 66.6 |
| PIC18LF4620-I/P | 4 | 5.46 | 21.84 | 5.46 | 21.84 |
| ATS100B-E | 4 | 0.36 | 1.44 | 0.36 | 1.44 |
| LR14XWA/BB | 6 | 1.33 | 7.98 | 1.33 | 7.98 |
| CR-123A | 2 | 2.49 | 4.98 | 2.49 | 4.98 |
| BH2CL | 3 | 1.92 | 5.76 | 1.92 | 5.76 |
| BH123A | 2 | 1.02 | 2.04 | 1.02 | 2.04 |
| B3F-1052 | 6 | 0.41 | 2.46 | 0 | 0 |
| Total |  |  | 122.06 |  | 115.12 |
|  |  |  |  |  |  |
| PCB | Cost for three |  |  |  |  |
| Master Post | 22.75 |  |  |  |  |
| Slave Post | 45 |  |  |  |  |
| Handheld Unit | 45 |  |  |  |  |
| Total PCB Cost | 112.75 |  |  |  |  |
|  |  |  |  |  |  |
| Extra Parts Bought | Quantity | Price Per Unit (Acquired) | Total Extra Parts Price |  |  |
| TRM - 418 - LT | 2 | 16.65 | 33.3 |  |  |
| TRM - 433 - LT | 2 | 16.65 | 33.3 |  |  |
| ANT - 433 - HETH | 2 | 1.12 | 2.24 |  |  |
| Total |  |  | 68.84 |  |  |
|  |  |  |  |  |  |
| Total Project Cost | 296.71 |  |  |  |  |
| Budget | 550 |  |  |  |  |

## Data Sheets: