WIRELESS COMPUTER CONTROLLED ROBOTICS USING THE PIC16F77 MICROCONTROLLER

Melonee Wise
Physics 397
Spring 2004

Purpose

The purpose of this project is to develop both a robot and the digital RF control using the PIC16F77 microcontroller. The specific goals of the project are:

1. A Finished Robot
2. PIC Chip Setup
    - Configure the assembler code properly for the PIC16F77 chip.
3. Robot Control Protocol
    - Drive motor direction
    - Drive motor speed control
    - Arm motor control
    - Global include file
4. Initialization
    - USART
    - PWM
5. The Main Program
    - Main function
    - Send/Receive function
    - Decode function
    - Drive motor control function
    - Arm motor control function
6. Hardware Design
    - H-bridges
    - RF Setup
7. Visual Basic Program
8. Conclusion and Recommendations

A Finished Robot

For this project the final robot, Zippy, was built (Figure 1 & Figure 2).

Figure 1 Final Zippy and partner Derek King

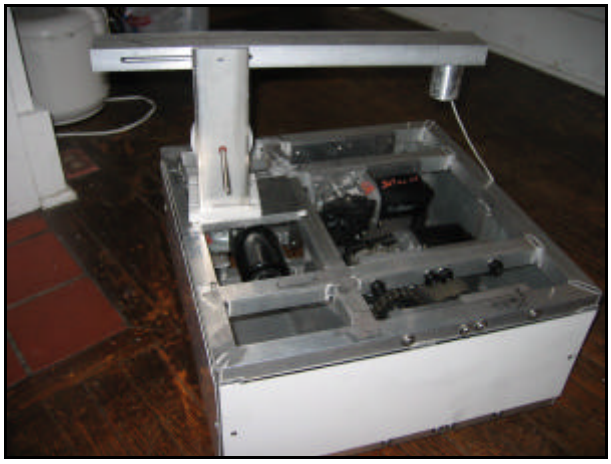

Figure 2 Topless Zippy without electronics

Zippy's main components consist of:

3 windshield wiper motors (purchased at Mack's Auto Recycling)
2 lawnmower wheels
4 gears and matching chains
1" square aluminum tubing (18" x 18" x 8" frame)
1 robotic arm (see picture)
1 Pitman motors (donated by the ECE department)
1 12V SLA battery (Panasonic LC-RD1217P)
1 yards of rope (for the magnetic lifter)
1 Tupperware container (to protect the antenna)
1 magnet
4 H-bridges

1 transceiver

Zippy is constructed using 1” square aluminum tubing welded together in to an 18” x 18” x 8” frame. Two drive motors are mounted in the interior to support plates and electrically isolated, typically windshield wiper motors have grounded casings and must be isolated to avoid a short across the frame. A drive gear is attached to each motor and connected by chain to the wheel sprocket. The wheel gear is attached to the wheel axel that drives the two push type lawnmower wheels. A third motor is mounted to the interior topside of the frame which directly drives the arm rotation. A small Pittman motor is used to drive the pulley of arm which raises and lowers the magnet.

Of the hardware used to construct Zippy only the electrical and computer components will be discussed within this report because the design of the robot is not focus of this project.

This is report will continue the work done in Fall 2003 in Physics 344 as part of my final project, the results of that project can be found at http://wug.physics.uiuc.edu/courses/phys344/Projects/Fall2003/Digital_Communication_PIC16F84A_Controller_Melonee_Wise_Fall2003.pdf.

Communication Software and Methods

1. PIC Setup

Before beginning the project, the necessary programs must be obtained. MPLAB can be downloaded for free from http://www.microchip.com and a programmer and programming software can obtained from http://ramseyelectronics.com. After the proper software and equipment are obtained the PIC chip must be correctly configured in MPLAB so that it can be programmed properly and functions properly when tested. Remember to always read the PIC data sheet before beginning any project.

First a project must be setup in MPLAB by creating a new project using the project wizard. The project wizard will step through selecting the proper device (PIC16F77), the proper language toolsuite (MPASM Assembler), and finally the project name and directory. Within the project the proper include and linker files must be added, the include files can be found in the MPLAB_IDE folder under disPIC_Tools/support/inc and the linker files in the MPLAB_IDE folder under MCHIP_Tools/Lkr. Finally a main source file must be created for executable code.

Next it is important to have the proper configuration; this sets the oscillator type, the watchdog timer, copy protection, and power up timer. For this project, the main source file is configured in the following manor,

__config _WDT_OFF & _PWRTE_ON & _HS_OSC & _CP_OFF,

this turns the watchdog timer off, the power up timer on, sets the oscillator to high speed, and turns copy protection off.

2. Movement Protocol

Before starting to create a program that controls Zippy a movement protocol was developed. The ports were also chosen for control. Some of the protocol listed was not fully implemented in the end do to time limitations.

Robot message protocol

0; 0UUU|ABCD = drive motor control (first byte)

ABCD=
0000 0 - set spool position
0001 1 - stop motors
0010 2 - pwm motors forward
0011 3 - pwm motors backward
0100 4 - pwm motors turn right
0101 5 - pwm motors turn left
0110 6 - change pwm speed
0110 7 - nothing
1000 8 - stepper stop
1001 9 - stepper set waitc
1010 10 - stepper forward
1011 11 - stepper reverse

1; 0XXX|XXXX = amount low value /right speed (second byte)
2; 0XXX|XXXX = amount high value /left speed (third byte)
3; 0ABC|DEFG = arm control (fourth byte)

String motor status:

A=0 - not moving
A=1 - moving
B=0 - moving up
B=1 - moving down

C=0 piece sensor no down
C=1 piece sensor hit
D=0 magnet sensor not hit
D=1 magnet sensor hit

String motor command:

A=0 don't do anything
A=1 take new command
B=0 – move up
C=0 stop at piece
C=1 stop at magnet
B=1 – move down
E = 0 arm not moving
F = 0 arm out
F = 1 arm in
E = 1 arm moving
F = 0 moving out
F = 1 moving in

Ports:

B0- right motor direction
0 = forward
1 = back
B1- left motor dir
0 = forward
1 = back
B2- string motor control1
B3- string motor control2
B4- arm motor control1
B5- arm motor control2

To simplify the code used to control Zippy a global include file, shown below, was made that contained many of the variables listed above along with simple macros for checking parity and moving the message sent.

GLOBAL INCLUDE

```
;global.inc

MSG_LEN     equ 4  ;number of bytes of data in a message (does
not include parity)

;for spi_flags
SPI_MSG_READY_FLAG      equ 1 ;message ready to be through SPI to
computer
CMD_MSG_READY_FLAG      equ 2 ;message ready to be sent through
usart to robot
SENDING_CMD_FLAG        equ 3       ;flag to remember if command
was
                                    ;being sent USART control

;for motor direction commands
DIR_STOP          equ 0
```

```
DIR_FORWARD       equ 2
DIR_BACKWARD      equ 3
DIR_RIGHT         equ 4
DIR_LEFT          equ 5

;for motor control
RMOTOR_PIN  equ 0
LMOTOR_PIN  equ 1

;calculates parity of msg, and returns with result in W
calc_parity macro the_msg
      movf the_msg, W
      local i=1
      while i<MSG_LEN
      xorwf (the_msg+i), W
i += 1
      endw
      endm

;copies one message from one place to another in the same bank
copy_msg macro from, to
      local i=0
      while i<MSG_LEN
      movf from+i, W
      movwf to+i
i=i+1
      endw
      endm

;copies one message from one place to another in the same bank
clear_msg macro the_msg
      local i=0
      while i<MSG_LEN
      clrf the_msg+i
i=i+1
      endw
      endm
```

3. Initialization

One of the biggest difficulties when using a larger PIC chip is initializing the registers so that the data is moved around correctly between the different memory banks. Additionally when using a chip with special functions, like PWM and USART, all of the registers must be configured properly for the chip to work in the expected manner.

First the USART was initialized using the following code and tested using hyper term. Three wires were soldered to the GRN, TX, and RX pins as shown below in Figure 3, and connected to an RS232 chip. To test if this had been done properly the transmitted data was received and retransmitted to the computer.

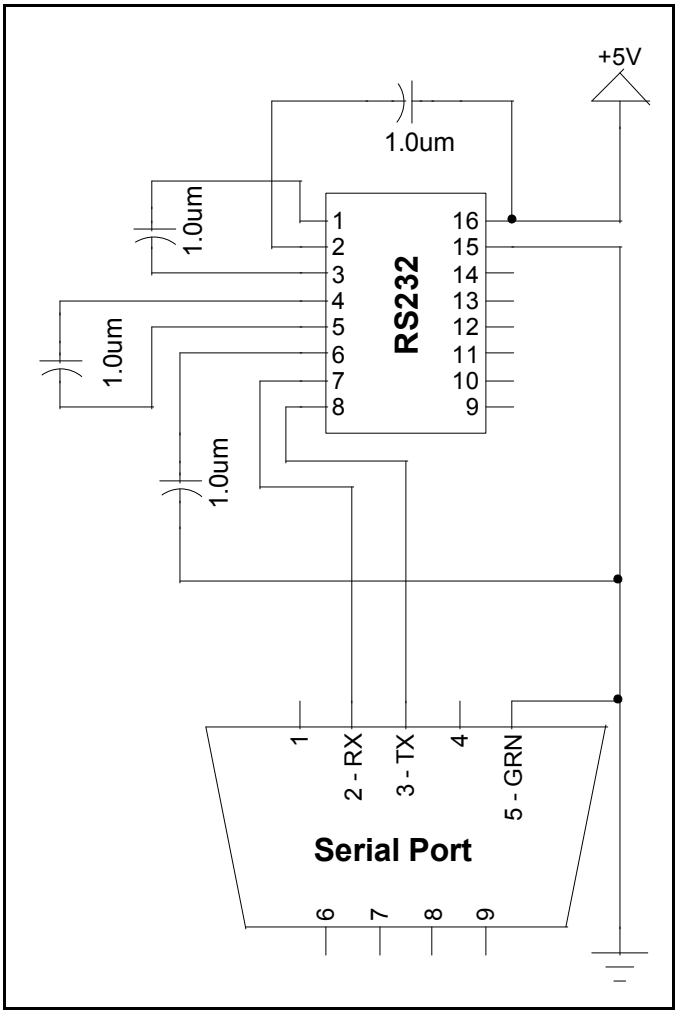

Figure 3 Serial Setup

USART INIT

```
      processor PIC16F77
      #include "P16F77.INC"

      global usart_init

CODE2 CODE

usart_init:

; SBRG 99h
; 3 gives a baud rate close to 19,231 with a 16MHz clock and
; BRGH=1
      banksel SPBRG
      movlw .51
      movwf SPBRG

; TXSTA 98h
      banksel TXSTA
```

```
;7 - CXRC - ? clock source select (does not matter 4 async)
;6 - TX9 - 0 8 bit transmission
;5 - TXEN - 1 enable transmit
;4 - SYNC - 0 async full
;3 - ? - ? blank
;2 - BRGH - 1 high speed baud
;1 - TRMT - 1 transmit is empty
;0 - TX9D - ? 9 bit for transmitting
      movlw B'00100110'
      movwf TXSTA

; RCSTA 18h
      banksel RCSTA
;7 - SPEN - 1 enable serial (overall)
;6 - RX9 - 0 select 8 bit receive (1=9bit, 0=8bit)
;5 - ? - ? don't care in async mode
;4 - CREN - 1 enable continuous recv
;3 - ? - ? nothing
;2 -  - 0 no framing error
;1 -  - 0 no overrun error
;0 -  - 0 9th bit of recv data
      movlw B'10010000'
      movwf RCSTA
      return

      end
```

Next the PWM was initialized using the following code and tested by calling a PWM function on the chip and looking at the pin outs with an oscilloscope to see if the output was correct.

PWM INIT

```
      processor PIC16F77
      #include "P16F77.INC"
      global pwm_init

CODE2 code

pwm_init:
;set the PWM period to about 1kHZ (16 MHz system clock)
;(PR2 is used by timer 2)
      banksel PR2       ;(92h)
      movlw 0xff
      movwf PR2

;set PWM 1&2 duty cycle to 0 to begin with
      banksel CCPR1L
      clrf CCPR1L
      bcf CCP1CON, CCP1X
      bcf CCP1CON, CCP1Y
      clrf CCPR2L
      bcf CCP2CON, CCP2X
      bcf CCP2CON, CCP2Y
```

```
;enable Timer2 and set prescale to 16
      banksel T2CON
      bsf T2CON, T2CKPS1      ;1x = 16x prescale
      bsf T2CON, TMR2ON ;1 = timer two is on

;put CCP 1&2 in PWM mode
;CCPxCON(3:0) = 11xx
      bsf CCP1CON, CCP1M3
      bsf CCP1CON, CCP1M2
      bsf CCP2CON, CCP1M3
      bsf CCP2CON, CCP1M2

;TRISC - set PORTC pin 1 & 2 in output mode
;so they can be used as pwm signals
      banksel TRISC
      bcf TRISC, 1            ;pin1 as output
      bcf TRISC, 2            ;pin2 as output
      return

      end
```

Some testing was also done using the SPI function on the chips. At first it looked like a good idea to have a second chip between the computer and the robot to do low level processing but this was not very effective because SPI and USART do not interface well. Figure 4 shows the setup for using SPI and the code below was used to initialize the SPI but was not used for controlling Zippy in the end.

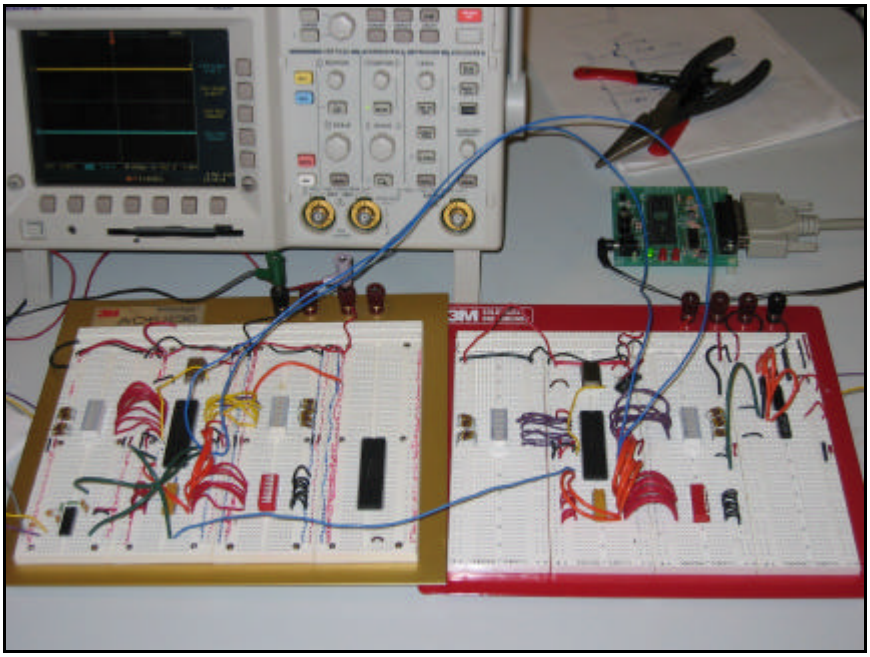

Figure 4 SPI (two chip interface) board setup

SPI INIT

```
      processor PIC16F77
```

```
      #include "P16F77.INC"

      global spi_master_init
      global spi_slave_init

CODE0 CODE

spi_master_init:
;5 - 0 SDO - 0 to enable output
;4 - 1 SDI - 1 to enable input
;3 - 0 SCK clock - output for SPI master mode
      banksel TRISC
      bcf TRISC, 5
      bsf TRISC, 4
      bcf TRISC, 3

;SPI setup
;SSPSTAT 94h
;7 - 0 SMP - sample in input middle of data output time
;6 - 1 CKE - set data trans for falling edge of clock when CKP=1
;5 - 0 I2C only
;4 - 0 I2C only
;3 - 0 I2C only
;2 - 0 I2C only
;1 - 0 I2C only
;0 - 0 BF - buffer status bit
      banksel SSPSTAT
      movlw B'01000000'
      movwf SSPSTAT

;SSPCON 14h
;7 - 0 WCOL - write collision flag
;6 - 0 SSPOV - receive overflow indicator
;5 - 1 SSPEN - enable sync serial port
;4 - 1 CKP - set idle clock state high
;3 - 0010 SSPM3:SSPM0 Mastah Mode
      banksel SSPCON
      movlw B'00110010'
      movwf SSPCON

      movf PORTD, W
      movwf SSPBUF

      return

spi_slave_init:
;5 - 0 SDO - 0 to enable output
;4 - 1 SDI - 1 to enable input
;3 - 1 SCK clock - input for SPI slave mode
      banksel TRISC
      bcf TRISC, 5
      bsf TRISC, 4
      bsf TRISC, 3

;SPI setup
;SSPSTAT 94h
```

```
;7 - 0 SMP - must be cleared for slave mode
;6 - 1 CKE - set data trans for falling edge of clock when CKP=1
;5 - 0 I2C only
;4 - 0 I2C only
;3 - 0 I2C only
;2 - 0 I2C only
;1 - 0 I2C only
;0 - 0 BF - buffer status bit
      banksel SSPSTAT
      movlw B'01000000'
      movwf SSPSTAT

;SSPCON 14h
;7 - 0 WCOL - write collision flag
;6 - 0 SSPOV - receive overflow indicator
;5 - 1 SSPEN - enable sync serial port
;4 - 1 CKP - set idle clock state high
;3 - 0101 SSPM3:SSPM0 Slave Mode -SS disabled
      banksel SSPCON
      movlw B'00110101'
      movwf SSPCON

      return

      end
```

4. The Main Program

The main robot control program was comprised of four different sub controls, the drive motor control, the send/receiver, the arm control, and the message decoder. One important thing to notice is the use of bank select, it is very important that the proper bank is selected before trying to jump to a function. Without selecting the proper bank the chip will malfunction. Also don't forget to clear the watchdog timer, if it is not cleared the chip will reset and act very strangely making it hard to debug.

THE MAIN FUNCTION

```
      __config (_PWRTE_ON & _WDT_ON & _HS_OSC)

      processor PIC16F77
      #include "P16F77.INC"

      extern      recv_mode_init, send_mode_init, sendrecv_init,
sendrecv_run
      extern      usart_init

      extern pwm_init

      extern motor_run, motor_init
      extern spool_init, spool_run

INT_FLAG    equ   0           ;flags interrupt occurrence
```

```
SHARED      udata
pclath_temp       res 1
status_temp       res 1
w_temp      res 1

REGS0       udata
this_bank   res 0
count1            res 1
count2            res 1
count3            res 1

STARTUP code
      goto init
      goto stop
      goto stop
      goto stop

;#################### INTERUPT ######################
isr:
      movwf w_temp      ;save context
      swapf STATUS, W
      clrf STATUS
      movwf status_temp
      movf PCLATH, w
      movwf pclath_temp

      banksel this_bank
      pagesel this_page
      ;was there a TIMER OVERFLOW int?
      btfss INTCON, T0IF
      goto not_timer_int
      bcf INTCON, T0IF
      incf count1, F          ;increment counters
      incf count2, F
      incf count3, F
not_timer_int:

      movwf pclath_temp       ;restore context
      movwf PCLATH
      swapf status_temp, W
      movwf STATUS
      swapf w_temp, F
      swapf w_temp, W
      retfie

CODE0 code
this_page:   ;for pagesel

;pic 16f77 initialization code
init:

;first disable all interrupts while initializing
      clrf INTCON

;option_reg 0x81, 0x181
```

```
;7 - 1 disable portB pull-ups
;6 - ? int edge
;5 - 0 tmr0 source (instruction clock)
;4 - ? tmr0 edge select
;3 - 1 prescale WDT
;2-0 - 000, 1:1 prescale for wdt
      banksel OPTION_REG
      movlw B'10001000'
      movwf OPTION_REG

;PIE1 - enable bit for peripheral interrupts
;7 - 0 don't enable parallel r/w interrupt
;6 - 0 don't enable A/D conversion interrupt
;5 - 0 disable USART receive enable
;4 - 0 disable USART transmit
;3 - 0 disable synchronous serial port enable
;2 - 0 disable cc1?
;1 - 0 disable timer 2
;0 - 0 disable timer 1
      banksel PIE1
      movlw B'00000000'
      movwf PIE1

;port A setup -
      banksel TRISA
      movlw B'00000000'       ;set porta to outputs
      movwf TRISA

;do stuff with ADCON1
      banksel ADCON1
      movlw 0x06        ;configure all PORTA pins as digital
inputs
      movwf ADCON1

;port B setup - make port B an output
      banksel TRISB
      clrf TRISB

;port C setup - setup for USART i/o
      banksel TRISC
      movlw B'11111111'
      movwf TRISC

;port D setup - mask port D input
      banksel TRISD
      movlw B'11111111'
      movwf TRISD

;port E setup - use as outputs
      banksel TRISE
      clrf TRISE

;initialize pwm
      pagesel pwm_init
      call pwm_init
```

```
;initialize usart
      pagesel usart_init
      call usart_init

;initialize usart snd/rcv protocol
      pagesel recv_mode_init  ;sendrecv_init
      call recv_mode_init           ;sendrecv_init

;initialize motor control
      pagesel motor_init
      call motor_init

;initialize spool motor control
      pagesel spool_init
      call spool_init

;reset bank and page
      pagesel this_page
      banksel this_bank
      bankisel this_bank

;INTCON 0x0b 0x8b 0x10b 0x18b
;7 - 1 enable global interrupts
;6 - 0 don't enable peripheral interrupts for now
;5 - 1 enable timer0 overflow interrupt
;4 - 0 don't enable rb0/int external interrupt
;3 - 0 don't enable rb port change interrupt
;2-0 - 000 ;interrupt flags
      movlw B'10100000'
      movwf INTCON

;output crap to ports a just to see if something is going on
      movlw B'11111111'
      movwf PORTA
      movwf PORTB
      movwf PORTE

;clear out timer counts
      banksel this_bank
      clrf count1
      clrf count2

;####################### MAIN ###################
main_loop:
      clrwdt

      movf count1, W
      sublw .4
      btfss STATUS, Z
      goto not_count1
      clrf count1
      pagesel sendrecv_run
      call sendrecv_run
      pagesel this_page
      banksel this_bank
      bankisel this_bank
```

```
not_count1:

      movf count2, W
      sublw .16
      btfss STATUS, Z
      goto not_count2
      clrf count2
      pagesel motor_run
      call motor_run
      pagesel spool_run
      call spool_run
      pagesel this_page
      banksel this_bank
      bankisel this_bank
not_count2:

      movf count3, W
      sublw .255
      btfss STATUS, Z
      goto not_count3
      clrf count3
      movlw B'00000001'
      xorwf PORTA, F
not_count3:

      goto main_loop

stop:
      goto stop

      END
```

The main function basically contains all of the port initializations, the interrupt service routine, and the basic call routine for sending and receiving data.

SEND/RECEIVE

```
      processor PIC16F77
      #include "P16F77.INC"

;Send receive message using USART
;use 0xff as a start/escape bit

      #include "global.inc"

DEBUG_PORTA             equ PORTA
IN_SEND_BIT             equ 3
IN_RECV_BIT             equ 2

;for flags
LAST_WAS_FF_FLAG equ 0  ;flags that last byte received was a 0xff
byte

FF_SEND_CYCLES    equ .50 ;number of cycles spent send 0xff bytes
before actual message is sent
```

```
RECV_CYCLES       equ .100 ;number of cycles to wait after
recieving a complete message

      ;for main
      global      recv_mode_init, send_mode_init, sendrecv_init,
sendrecv_run

      ;for control file
      global      state, msg, waitc
      global      send_start, recv_start

      ;from control file
      extern send_complete_f, recv_complete_f, recv_error_f

REGS0 UDATA
this_bank   res 0
waitc       res 1       ;wait count
flags       res 1       ;marks mode that transceiver should be in
bytec       res 1       ;byte count
btemp       res 1       ;byte temporary value for sending or
recieving from
state       res 1       ;address of function to run next
msg         res MSG_LEN ;temporary space used when send/recv new
message
parity      res 1       ;parity must be right after message


CODE3 code

sendrecv_run:
      banksel this_bank
      bankisel this_bank
      ;indirect goto
      movf state, W
      movwf PCL   ;goto state

;###### SEND FUNCTIONS ###### ###### ###### ###### ######

send_start:
      ;start sending out start bit
      bsf DEBUG_PORTA, IN_SEND_BIT

      ;calculate parity for message
      calc_parity msg
      xorlw 0x80
      movwf parity
      movlw MSG_LEN + 1  ;MSG_LEN+1 to send parity
      movwf bytec
      movlw FF_SEND_CYCLES
      movwf waitc
      movlw send_byte_ff
      movwf state

;send 0xff's for a while to allow receiver to lock
send_byte_ff:
      btfss PIR1, TXIF
```

```
      goto send_byte_ff_end ;is usart ready to send another byte
      movlw 0xff
      movwf TXREG
send_byte_ff_end:
      decfsz waitc, F
      return
      movlw send_byte
      movwf state

;send actual message
send_byte:
      btfss PIR1, TXIF  ;is usart ready to send another byte
      return
      decf bytec, W
      addlw msg
      movwf FSR
      movf INDF, W
      movwf TXREG
      decfsz bytec, F
      return

send_end:
      bcf DEBUG_PORTA, IN_SEND_BIT
      pagesel send_complete_f
      goto send_complete_f

;###### ###### ###### ###### ###### ###### ###### ######
;###### RECEIVE FUNCTIONS ###### ###### ###### ###### ######
; receive uses 0xff as the start bit for transmission
; if the transmission need to send 0xff it should escape it
; with another 0xff

recv_start:
      bsf DEBUG_PORTA, IN_RECV_BIT

      ;set transceiver to receive mode here and wait

      ;setup to wait to receive first byte of data
      bcf RCSTA, CREN   ;reset usart rcv to clear any errors or
      any queued data
      bsf RCSTA, CREN
      clrf bytec        ;start out cycle length as 0
      movlw RECV_CYCLES
      movwf waitc
      movlw recv_byte   ;set up recv
      movwf state

recv_byte:
      ;check for new data
      btfss PIR1, RCIF
      goto recv_byte_end      ;there is no new data
      ;store new data
      movf RCREG, W
      movwf btemp

      ;check to see if new value was 0xff
```

```
        comf btemp, W
        btfss STATUS, Z
        goto test_for_start_condition

        ;if new byte was byte was 0xff check old 0xff flags
        btfsc flags, LAST_WAS_FF_FLAG
        goto try_recv_store_byte      ;add new 0xff value, clear
        flag

        ;set flag don't do anything for now
        bsf flags, LAST_WAS_FF_FLAG
        goto recv_byte_end

test_for_start_condition:     ;if new value is not 0xff check to
        see if old value was
        btfss flags, LAST_WAS_FF_FLAG
        goto try_recv_store_byte

        ;if old byte was 0xff but new byte is not, restart message
        and store new byte
        movlw MSG_LEN + 1  ;MSG_LEN+1 to recv parity
        movwf bytec

;if bytec is already 0 don't store byte
try_recv_store_byte:
        bcf flags, LAST_WAS_FF_FLAG
        movf bytec, W
        btfsc STATUS, Z
        goto recv_byte_end

store_recv_byte:  ;store new byte in msg
        decf bytec, W     ;put new data into msg
        addlw msg
        movwf FSR
        movf btemp, W
        movwf INDF
        decfsz bytec, F
        goto recv_byte_end  ;message is not complete yet

        ;check parity of received message
        calc_parity msg
        xorwf parity, W
        btfss STATUS, Z   ;a correct parity will result with W=0
        goto recv_byte_end  ;if parity is in error just keep
        reading

        ;if partity is correct, a complete message has been receive
        bcf DEBUG_PORTA, IN_RECV_BIT
        pagesel recv_complete_f
        goto recv_complete_f

recv_byte_end:
        ;check to see if time has run out
        decfsz waitc, F   ;check to see if message read operation
timed out
        return
```

```
      ;there was no valid message receive in the allotted amount
of time
      bcf DEBUG_PORTA, IN_RECV_BIT
      pagesel recv_error_f
      goto recv_error_f

;###### ###### ###### ###### ###### ###### ###### ######
;###### INIT FUNCTIONS ###### ###### ###### ###### ######
recv_mode_init:
      banksel this_bank
      ;set start state
      movlw recv_start
      movwf state
      goto sendrecv_init

send_mode_init:
      banksel this_bank
      ;start in send state for now
      movlw send_start
      movwf state

;general init stuff
sendrecv_init:
      clrf flags
      movf PORTD, W
      movwf TXREG       ;send initial value to set trx interrupt
flags
      return

      END
```

The send/receive function does exactly what is expected, receives data from the computer and replies with the robots current state. To receive the chip sets the transceiver in receive mode and waits. Then the function looks for the start byte of 0xff, once it has received the start byte it starts storing the message to memory. Once the entire message is stored and the parity is checked, the decoding function, shown below, is called. One thing to be careful of is the use of jump tables like the one in the decoding function, because it adds to the PCL. If the jump table occurs at the end of a bank it could cause a jump to someplace unintended.

DECODING FUNCTION

```
      processor PIC16F77
      #include "P16F77.INC"

      #include "global.inc"

      ;from usart
      extern state, msg
      extern send_start, recv_start
```

```
	;for usart
	global send_complete_f, recv_complete_f, recv_error_f

	;for motor control
	extern motor_dir, dist_l, dist_h, r_speed, l_speed

	;for stepper control
	;extern stepper_dist_l, stepper_dist_h, stepper_dir

	extern p_flags, p_waitc

;SHARED udata
;share_msg  res   MSG_LEN          ;used   to   copy   messages
between banks

CODE0 code

;send complete should set up trx chip to wait for a new incoming
message
send_complete_f:
	banksel state
	movlw recv_start
	movwf state
	return

;if there was an error recieving a msg, try recieving again
recv_error_f:
	;first check if it was a control messgage that was
	;not responded to...
	banksel state
	bcf PORTA, 1
	;goto recv_complete_f
	movlw recv_start
	movwf state
	return

;look at decode msg and set up to send reply
recv_complete_f:
	banksel msg
	bsf PORTA, 1
	;decode message
	movf msg, W
	andlw 0x07
	addwf PCL, F
	;jump table
	goto set_spool              ;0
	goto set_motor_dir          ;1
	goto set_motor_dir          ;2
	goto set_motor_dir          ;3
	goto set_motor_dir          ;4
	goto set_motor_dir          ;5
	goto set_motor_speed        ;6
	goto finish_message         ;7

set_spool:
	movf msg+3, W
```

```
        andlw 0x07
        movwf p_flags
        movf msg+1, W
        movwf p_waitc
        goto finish_message

;set a new motor direction and distance
set_motor_dir:
        ;set new direction
        movf msg, W
        andlw 0x07
        movwf motor_dir
        ;extract new distance
        movf msg+1, W
        movwf dist_l
        movf msg+2, W
        movwf dist_h
        goto finish_message

;msg[1] = right motor pwm
;msg[2] = left motor pwm
set_motor_speed:
        movf msg+1, W
        movwf r_speed
        movf msg+2, W
        movwf l_speed
        goto finish_message

finish_message:
        ;always reply with current state of robot
        ;no matter what message gets sent
        movf PORTD, W
        movwf msg+3

        ;fill in distance
        movf dist_l, W
        movwf msg+1
        movf dist_h, W
        movwf msg+2

        clrf msg
        movlw send_start
        movwf state
        return

        end
```

Once the message is decoded it then calls one of two functions to control either the drive motors or the arm motors. Figures 5 & 6 show the wiring diagram for the PIC chip with the direction line and PWM line control for the drive motors.

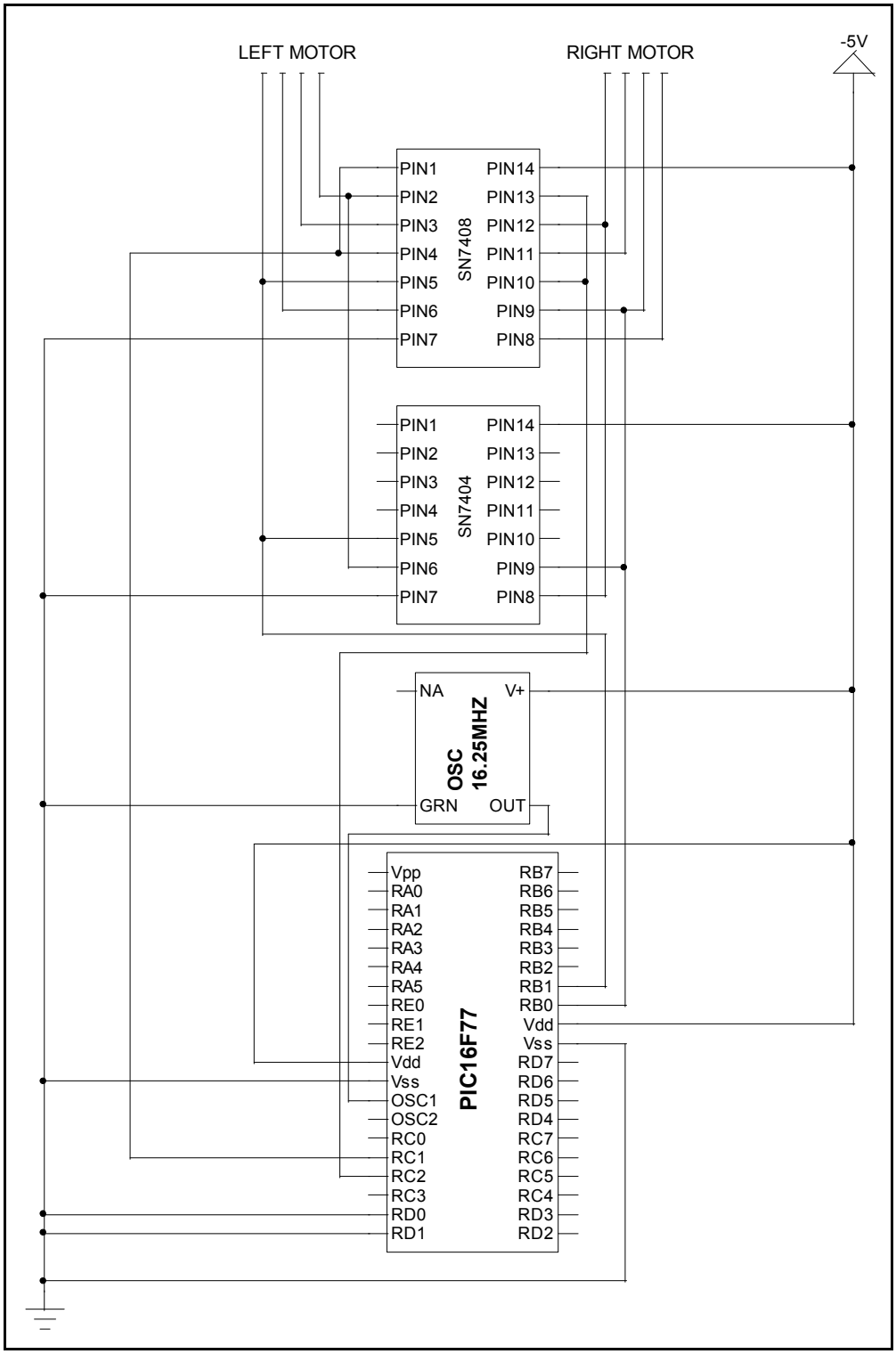

Figure 5 Wiring for direction logic

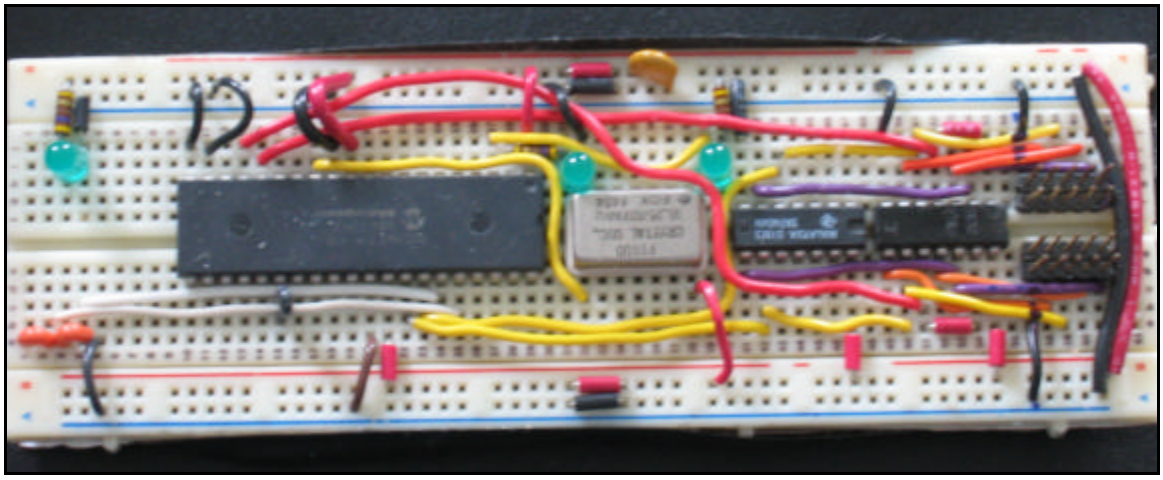

Figure 6 Actual Wiring

DRIVE MOTOR CONTROL FUNCTION

```
      processor PIC16F77
      #include "P16F77.INC"

      #include "global.inc"

      ;for usart
      global motor_dir, dist_l, dist_h, r_speed, l_speed
      global motor_run, motor_init

REGS0 udata
this_bank         res 1
motor_dir_old     res 1
motor_dir         res 1
      ;0 = stopped
      ;1 = ?
      ;2 = going forward
      ;3 = going backward
      ;4 = right
      ;5 = left
motor_waitc res 1 ;waitc count before going in next direction
dist_l      res 1
dist_h      res 1
r_speed     res 1
l_speed     res 1

PROG1 code

motor_init:
      banksel this_bank
      movlw 1
      movwf motor_waitc
      clrf dist_l
      clrf dist_h
      clrf motor_dir
      clrf motor_dir_old
      movlw 0x7f
```

```
        movwf r_speed
        movwf l_speed
        return

;check for motor control change
motor_run:
        banksel this_bank
        ;do not update motor until wait is up
        decfsz motor_waitc, F
        return
        incf motor_waitc, F

        ;see if motor direction changed
        movf motor_dir, W
        subwf motor_dir_old, W
        btfss STATUS, Z
        goto change_motor_dir

        movf motor_dir_old, W
        andlw 0x07
        addwf PCL, F
        goto motor_stop                 ;0
        goto motor_stop                 ;1
        goto motor_forward              ;2
        goto motor_backward             ;3
        goto motor_right                ;4
        goto motor_left                 ;5
        goto motor_stop                 ;6
        goto motor_stop                 ;7

motor_forward:
        bcf PORTB, RMOTOR_PIN
        bcf PORTB, LMOTOR_PIN
        goto motor_speed_check

motor_backward:
        bsf PORTB, RMOTOR_PIN
        bsf PORTB, LMOTOR_PIN
        goto motor_speed_check

motor_right:
        bsf PORTB, RMOTOR_PIN
        bcf PORTB, LMOTOR_PIN
        goto motor_speed_check

motor_left:
        bcf PORTB, RMOTOR_PIN
        bsf PORTB, LMOTOR_PIN
        goto motor_speed_check
        ;set pins based on motor direction

motor_speed_check:
        ;update motor speed here
        movf r_speed, W
        movwf CCPR1L
        movf l_speed, W
```

```
        movwf CCPR2L

        ;check distance count
        decfsz dist_l, F
        return
        decfsz dist_h, F
        return
        ;stop motors because movement is done

motor_stop:
        clrf motor_dir

change_motor_dir:
        ;stop motors and set wait before they start again
        movf motor_dir, W
        movwf motor_dir_old
        clrf CCPR1L
        clrf CCPR2L
        movlw 10
        movwf motor_waitc
        return

        end
```

This function uses a jump table to jump to the correct function to direct Zippy. A build in "distance" or more correctly time is used to stop the robot from running infinitely if the chip malfunctions.

ARM MOTOR CONTROL FUNCTION

```
        processor PIC16F77
        #include "P16F77.INC"

        #include "global.inc"

        ;for usart
        global p_flags, p_waitc
        global spool_init, spool_run

P_MOVE_FLAG       equ 2
P_DIR_FLAG        equ 1
P_STOP_FLAG       equ 0

REGS0       udata
this_bank   res 0
p_flags     res1  ;flags for deciding what to do with pulley
p_waitc     res 1 ;wait for lowering pulley

PROG1 code

spool_init:
        banksel this_bank
        clrf p_flags
        clrf p_waitc
        return
```

```
;check for motor control change
spool_run:
      banksel this_bank

      ;is the pulley supposed to be moving?
      btfss p_flags, P_MOVE_FLAG
      goto stop

      ;check count down
      decfsz p_waitc, F
      goto no_stop
      goto stop

no_stop:
      ;which direction is the pulley moving in?
      btfss p_flags, P_DIR_FLAG
      goto move_up

      ;pulley is moving down - check count
      bcf PORTB, 2
      bsf PORTB, 3
      return

move_up:
      bcf PORTB, 3
      bsf PORTB, 2
      ;stop no matter what, when switch 1 hits
      btfsc PORTD, 0
      goto stop

      ;if flag is set for piece stop then stop there
      btfss p_flags, P_STOP_FLAG
      return
      btfss PORTD, 0
      return

stop:
      ;stop pulley movement
      bcf p_flags, P_MOVE_FLAG
      bcf PORTB, 2
      bcf PORTB, 3
      return

      end
```

Control Hardware and Methods

1. H-bridges

Zippy was controlled using three H-bridges, the rotation motor for Zippy's arm was not implemented because of time constraints but a fourth H-bridge can be added and controlled using the code developed above. The H-bridges used for the

finished Zippy are very similar to the ones used in the prototype. High slew rate op-amps were added to limit the amount of time spent in the transition state of the mosfet preventing shoot thru current.

The board layouts were created using EASYTRAX, this software can be obtained from http://www.ece.uiuc.edu/eshop/pcbdesign/. This website also covers where and how to get PCB board cut on campus. Figure 7 shows a built up PCB board used for Zippy's motor control.

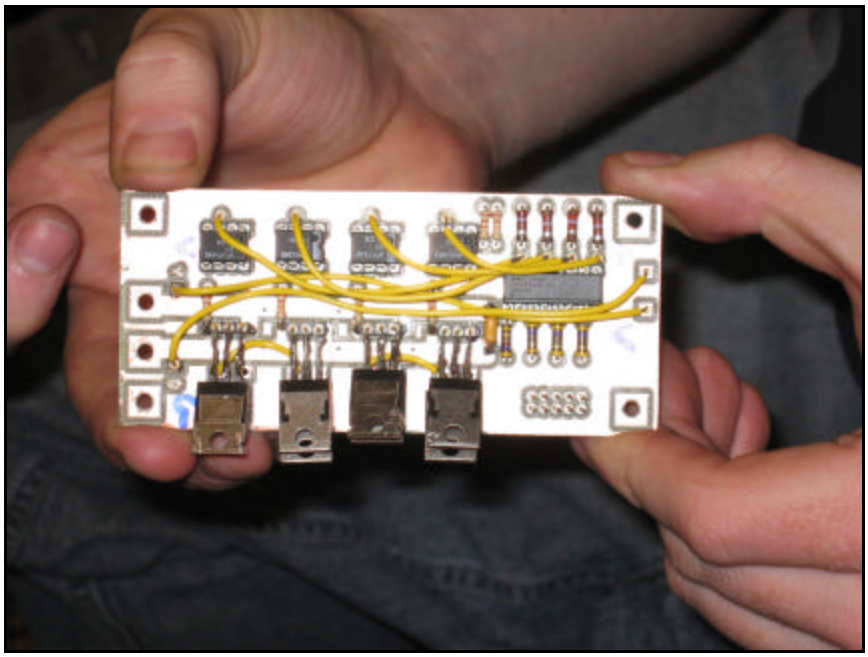

Figure 7 H-bridge board used to control Zippy

As can be seen in the picture two mosfets were piggybacked on top of each other to increase the current load possible for the mosfet. This also helped to dissipate some of the heat that is created from the imperfect switching of the mosfets. Figure 8 shows the wring layout of the above PCB board.

These boards were stacked together in a project box and connected to power and the batteries. Figure 9 shows the boards together in the project box, the project box was used to shield RF from the noise.

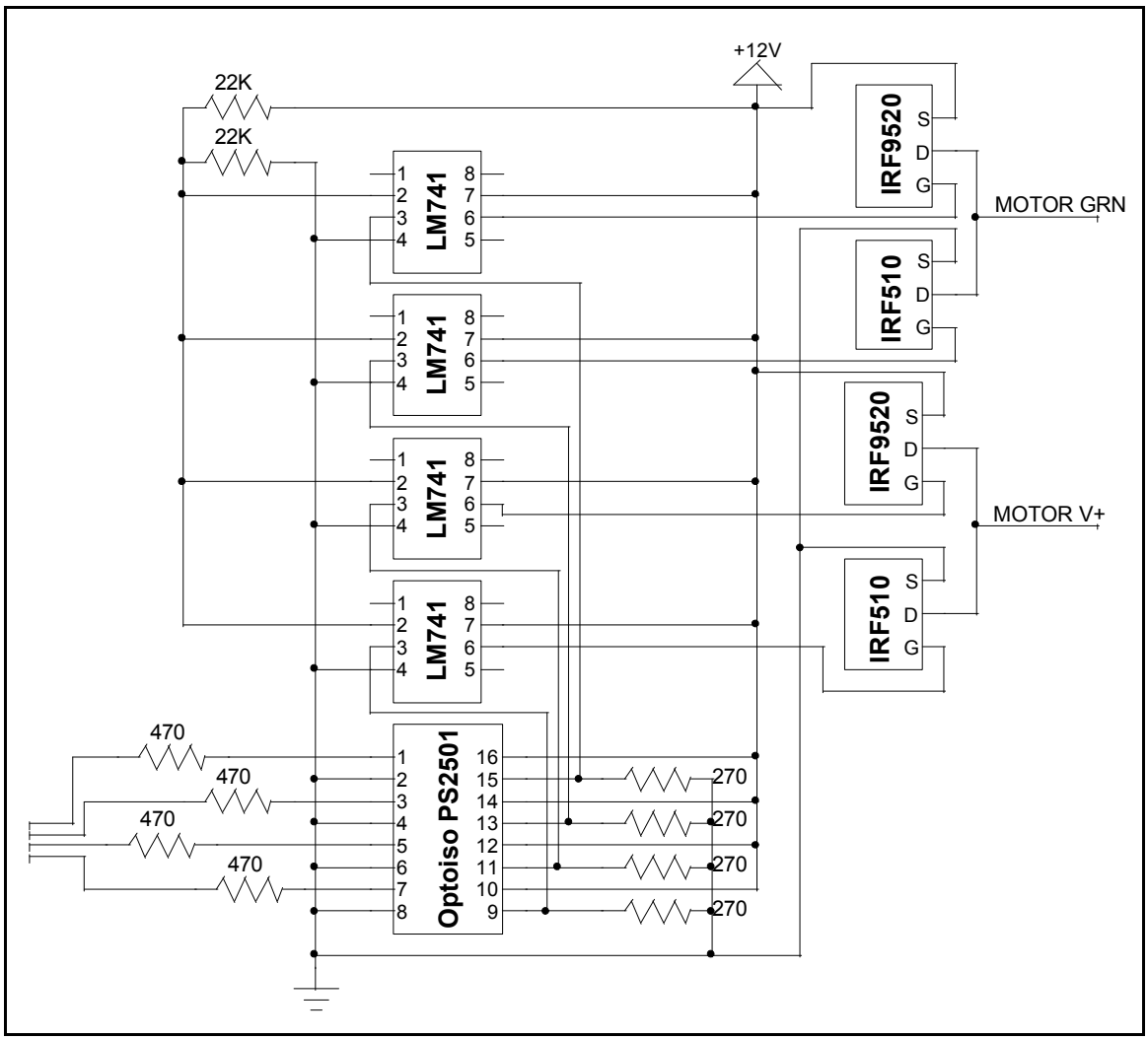

Figure 8 Wiring Diagram

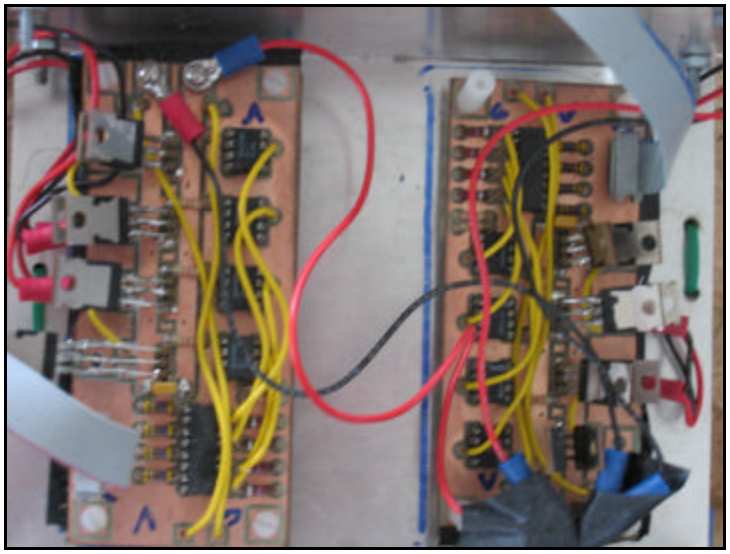

Figure 9 Project box with PCB boards

2. RF Setup

The RF transceivers were setup to take serial input from the computer and transmit it to the PIC chip which controlled Zippy. Figure 10 shows the RF wiring diagram for the transmit side connected to the computer.

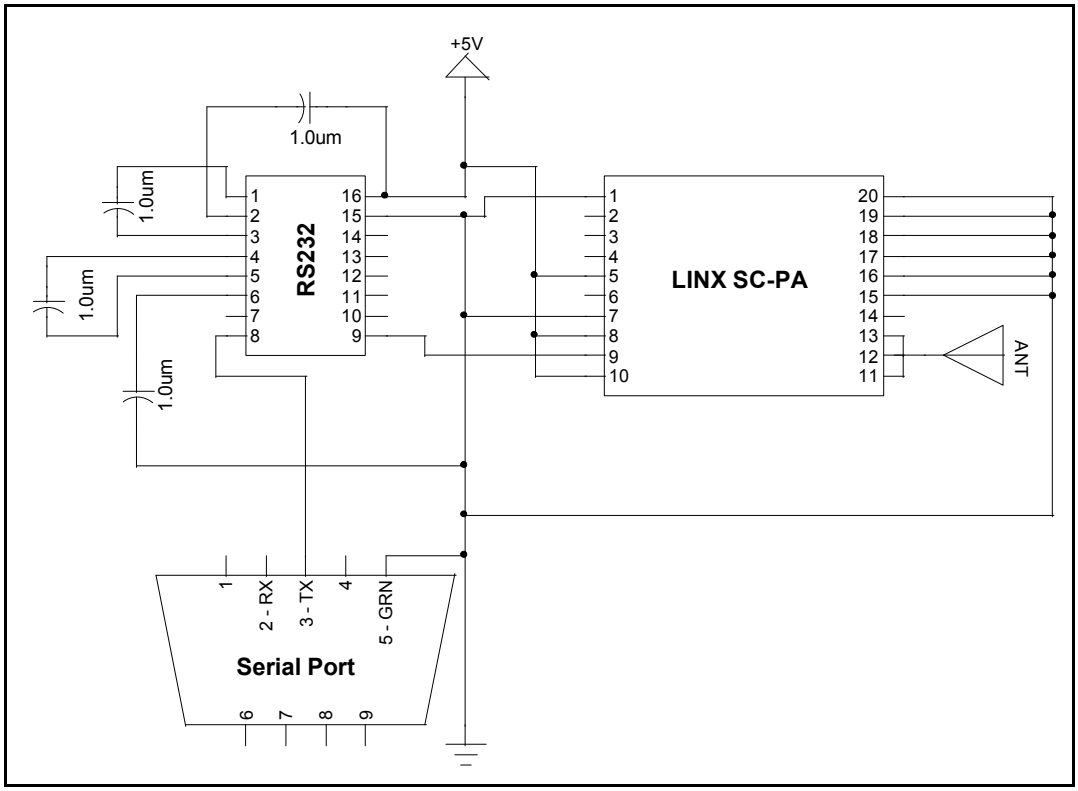

Figure 10 Transmitter Setup

Figure 11 shows the wiring diagram for the receiving end.

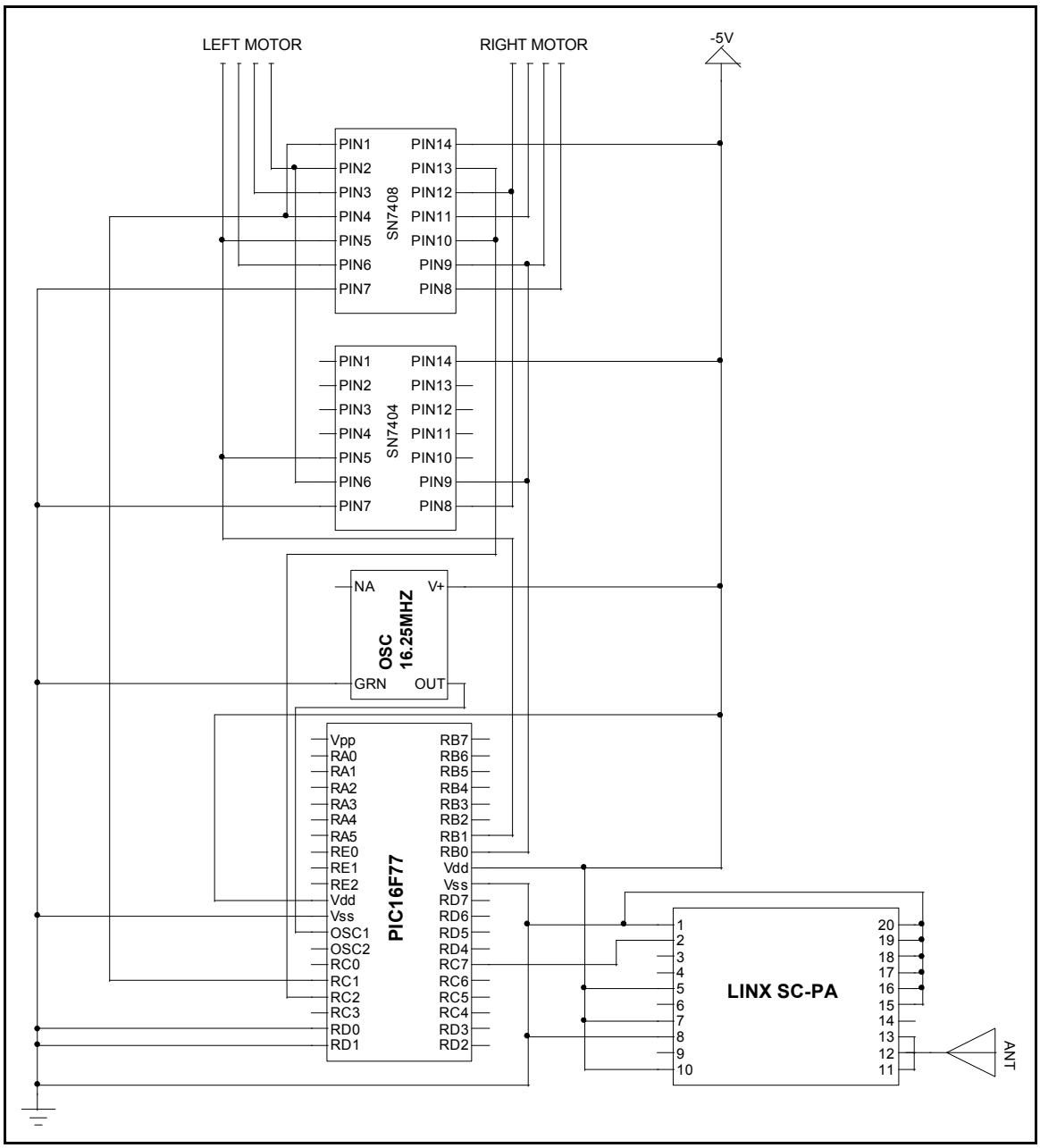

Figure 11 Receiver Setup

Visual Basic Program

The following code was used to drive zippy around using a standard USB joystick.

```
VERSION 5.00
Object = "{648A5603-2C6E-101B-82B6-000000000014}#1.1#0"; "MSCOMM32.OCX"
Begin VB.Form main_form
   Caption         =   "Robot Control"
   ClientHeight    =   9225
```

```
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   9570
   LinkTopic       =   "Form1"
   ScaleHeight     =   9225
   ScaleWidth      =   9570
   StartUpPosition =   3  'Windows Default
   Begin VB.Frame button
      Caption         =   "Button Control"
      Height          =   4455
      Left            =   2640
      TabIndex        =   16
      Top             =   4560
      Width           =   6375
      Begin VB.CommandButton forward
         Caption         =   "forward"
         Height          =   855
         Left            =   1320
         TabIndex        =   31
         Top             =   360
         Width           =   855
      End
      Begin VB.CommandButton left
         Caption         =   "left"
         Height          =   855
         Left            =   480
         TabIndex        =   30
         Top             =   1200
         Width           =   855
      End
      Begin VB.CommandButton right
         Caption         =   "right"
         Height          =   855
         Left            =   2160
         TabIndex        =   29
         Top             =   1200
         Width           =   855
      End
      Begin VB.CommandButton back
         Caption         =   "back"
         Height          =   855
         Left            =   1320
         TabIndex        =   28
         Top             =   2040
         Width           =   855
      End
      Begin VB.CommandButton stop
         Caption         =   "stop"
         Height          =   855
         Left            =   1320
         TabIndex        =   27
         Top             =   1200
         Width           =   855
      End
      Begin VB.CommandButton speed
         Caption         =   "Set Speed"
```

```
   Height          =   615
   Left            =   3600
   TabIndex        =   26
   Top             =   360
   Width           =   735
End
Begin VB.TextBox rf_speed
   Height          =   375
   Left            =   3600
   TabIndex        =   25
   Text            =   "Text1"
   Top             =   1200
   Width           =   1215
End
Begin VB.TextBox rb_speed
   Height          =   375
   Left            =   3600
   TabIndex        =   24
   Text            =   "Text1"
   Top             =   1680
   Width           =   1215
End
Begin VB.TextBox lf_speed
   Height          =   375
   Left            =   3600
   TabIndex        =   23
   Text            =   "Text1"
   Top             =   2160
   Width           =   1215
End
Begin VB.TextBox lb_speed
   Height          =   375
   Left            =   3600
   TabIndex        =   22
   Text            =   "Text1"
   Top             =   2640
   Width           =   1215
End
Begin VB.CommandButton up
   Caption         =   "up"
   Height          =   495
   Left            =   360
   TabIndex        =   21
   Top             =   3000
   Width           =   855
End
Begin VB.CommandButton down
   Caption         =   "down"
   Height          =   495
   Left            =   360
   TabIndex        =   20
   Top             =   3600
   Width           =   855
End
Begin VB.TextBox spool_dist
   Height          =   495
```

```
      Left            =   1320
      TabIndex        =   19
      Text            =   "Text1"
      Top             =   3600
      Width           =   975
   End
   Begin VB.CommandButton pstop
      Caption         =   "stop"
      Height          =   495
      Left            =   2400
      TabIndex        =   18
      Top             =   3000
      Width           =   975
   End
   Begin VB.CommandButton up2mag
      Caption         =   "up2mag"
      Height          =   495
      Left            =   1320
      TabIndex        =   17
      Top             =   3000
      Width           =   975
   End
   Begin VB.Label Label1
      Caption         =   "Right Forward"
      Height          =   375
      Left            =   4920
      TabIndex        =   35
      Top             =   1200
      Width           =   1095
   End
   Begin VB.Label Label2
      Caption         =   "Right Back"
      Height          =   375
      Left            =   4920
      TabIndex        =   34
      Top             =   1680
      Width           =   1095
   End
   Begin VB.Label Label3
      Caption         =   "Left Forward"
      Height          =   375
      Left            =   4920
      TabIndex        =   33
      Top             =   2160
      Width           =   1095
   End
   Begin VB.Label Label4
      Caption         =   "Left Back"
      Height          =   375
      Left            =   4920
      TabIndex        =   32
      Top             =   2640
      Width           =   1095
   End
End
Begin VB.Frame joystick
```

```
Caption         =   "Joystick"
Height          =   4335
Left            =   2640
TabIndex        =   4
Top             =   120
Width           =   6375
Begin VB.TextBox joy_rvel
   Height          =   375
   Left            =   2400
   TabIndex        =   37
   Text            =   "Right Vel"
   Top             =   3600
   Width           =   855
End
Begin VB.TextBox joy_lvel
   Height          =   375
   Left            =   1440
   TabIndex        =   36
   Text            =   "Left Vel"
   Top             =   3600
   Width           =   855
End
Begin VB.TextBox joy_y
   Height          =   375
   Left            =   1560
   TabIndex        =   15
   Text            =   "Y"
   Top             =   2160
   Width           =   735
End
Begin VB.TextBox joy_x_center
   Height          =   375
   Left            =   2400
   TabIndex        =   14
   Text            =   "X MID"
   Top             =   1680
   Width           =   735
End
Begin VB.TextBox joy_x
   Height          =   375
   Left            =   1560
   TabIndex        =   13
   Text            =   "X"
   Top             =   1680
   Width           =   735
End
Begin VB.TextBox joy_y_max
   Height          =   375
   Left            =   2040
   TabIndex        =   12
   Text            =   "YMAX"
   Top             =   2640
   Width           =   735
End
Begin VB.TextBox joy_num
   Height          =   375
```

```
      Left            =   3600
      TabIndex        =   10
      Text            =   "X"
      Top             =   600
      Width           =   375
   End
   Begin VB.CommandButton joy_start
      Caption         =   "Initialize Joystick"
      Height          =   375
      Left            =   1080
      TabIndex        =   9
      Top             =   600
      Width           =   1455
   End
   Begin VB.TextBox joy_y_center
      Height          =   375
      Left            =   2400
      TabIndex        =   8
      Text            =   "Y MID"
      Top             =   2160
      Width           =   735
   End
   Begin VB.TextBox joy_x_max
      Height          =   375
      Left            =   3240
      TabIndex        =   7
      Text            =   "XMAX"
      Top             =   1920
      Width           =   735
   End
   Begin VB.TextBox joy_x_min
      Height          =   375
      Left            =   720
      TabIndex        =   6
      Text            =   "XMIN"
      Top             =   1920
      Width           =   735
   End
   Begin VB.TextBox joy_y_min
      Height          =   375
      Left            =   2040
      TabIndex        =   5
      Text            =   "YMIN"
      Top             =   1200
      Width           =   735
   End
   Begin VB.Label Label5
      Caption         =   "Joystick #"
      Height          =   255
      Left            =   2760
      TabIndex        =   11
      Top             =   720
      Width           =   855
   End
End
Begin VB.CommandButton quit
```

```
      Caption         =   "quit"
      Height          =   615
      Left            =   120
      TabIndex        =   2
      Top             =   1680
      Width           =   615
   End
   Begin VB.CommandButton run
      Caption         =   "run"
      Height          =   615
      Left            =   120
      TabIndex        =   0
      Top             =   840
      Width           =   615
   End
   Begin MSCommLib.MSComm MSComm1
      Left            =   120
      Top             =   120
      _ExtentX        =   1005
      _ExtentY        =   1005
      _Version        =   393216
      DTREnable       =   0   'False
      OutBufferSize   =   5
      BaudRate        =   19200
   End
   Begin VB.Label runinfo
      Caption         =   "runinfo"
      Height          =   375
      Left            =   1200
      TabIndex        =   3
      Top             =   960
      Width           =   1095
   End
   Begin VB.Label info
      Caption         =   "info"
      Height          =   375
      Left            =   1200
      TabIndex        =   1
      Top             =   1800
      Width           =   1095
   End
End
Attribute VB_Name = "main_form"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim f As Boolean
Dim r As Boolean
Dim l As Boolean
Dim b As Boolean
Dim upb As Boolean
Dim downb As Boolean
Dim upup As Boolean
Dim quitflag As Boolean
```

```
Const USE_BUTTONS = 0
Const USE_JOYSTICK = 1
Dim input_type As Integer



Option Explicit
Private Declare Function joyGetDevCaps Lib "winmm.dll" Alias
"joyGetDevCapsA" (ByVal id As Long, lpCaps As joycaps, ByVal uSize As
Long) As Long
Private Declare Function joyGetPos Lib "winmm.dll" (ByVal uJoyID As
Long, pji As JoyInfo) As Long

Const MAXPNAMELEN = 32

Private Type joycaps
        wMid As Integer
        wPid As Integer
        szPname As String * MAXPNAMELEN
        wXmin As Long
        wXmax As Long
        wYmin As Long
        wYmax As Long
        wZmin As Long
        wZmax As Long
        wNumButtons As Long
        wPeriodMin As Long
        wPeriodMax As Long
End Type

Private Type JoyInfo
        wXpos As Long
        wYpos As Long
        wZpos As Long
        wButtons As Long
End Type

'Joystick error codes and return values
Const JOYERR_NOERROR = 0
Const JOYERR_BASE As Long = 160
Const JOYERR_UNPLUGGED As Long = (JOYERR_BASE + 7)
Const MMSYSERR_BASE As Long = 0
Const MMSYSERR_NODRIVER As Long = (MMSYSERR_BASE + 6)
Const MMSYSERR_INVALPARAM As Long = (MMSYSERR_BASE + 11)
Const joystick1 As Long = &H0
Const JOYSTICK2 As Long = &H1
Const JOY_BUTTON2 = &H2
Const JOY_BUTTON1 = &H1

Private Type joystick
    x_max As Long
    y_max As Long
    x_min As Long
    y_min As Long
    x_center As Long
    y_center As Long
```

```
End Type

Dim js As joystick

Sub InitJoystick()
    Dim rt As Long
    Dim JoyInformation As JoyInfo
    Dim JoyStickCaps As joycaps

    'set joystick range
    joyGetDevCaps joystick1, JoyStickCaps, Len(JoyStickCaps)
    With JoyStickCaps
        'js.x_max = .wXmax
        'js.x_min = .wXmin
        'js.y_max = .wYmax
        'js.y_min = .wYmin
        joy_x_max.Text = .wXmax
        joy_x_min.Text = .wXmin
        joy_y_max.Text = .wYmax
        joy_y_min.Text = .wYmin
    End With

    'center joystick
    joyGetPos joystick1, JoyInformation
    'js.x_center = JoyInformation.wXpos
    'js.y_center = JoyInformation.wYpos
    joy_x_center.Text = JoyInformation.wXpos
    joy_y_center.Text = JoyInformation.wYpos

    input_type = USE_JOYSTICK

End Sub

Sub SendCommand(command As Integer, dist_l As Integer, dist_h As
Integer, other As Integer)
    Dim parity As Integer
    Dim msg As String
    Dim index As Integer

    If MSComm1.PortOpen = False Then
        MSComm1.PortOpen = True
    End If

    'send message start byte
    MSComm1.Output = Chr(255) + Chr(127) + Chr(255)

    'calculate message parity
    parity = (command Xor dist_l Xor dist_h Xor other)

    msg = Chr(parity) + Chr(other) + Chr(dist_h) + Chr(dist_l) +
Chr(command)

    For index = 1 To Len(msg)
        MSComm1.Output = Mid(msg, index, 1)
        If Mid(msg, index, 1) = Chr(255) Then
            MSComm1.Output = Chr(255)
```

```
        End If
    Next index

End Sub

Sub GetButtons()
    'Uses the position of the form buttons
    'to control the robot's movements
    If f = True Then
        Call SendCommand(2, 80, 2, 0)
        info.Caption = "forward"
    ElseIf b = True Then
        Call SendCommand(3, 80, 2, 0)
        info.Caption = "backward"
    ElseIf r = True Then
        Call SendCommand(4, 80, 2, 0)
        info.Caption = "right"
    ElseIf l = True Then
        Call SendCommand(5, 80, 2, 0)
        info.Caption = "left"
    ElseIf upb = True Then
                    If Not IsNumeric(spool_dist.Text) Then
                        spool_dist.Text = "127"
                    End If
                    Call SendCommand(0, CByte(spool_dist.Text), 0, 4)
                    info.Caption = "up"
    ElseIf upup = True Then
                    If Not IsNumeric(spool_dist.Text) Then
                        spool_dist.Text = "127"
                    End If
                    Call SendCommand(0, CByte(spool_dist.Text), 0, 5)
                    info.Caption = "upup"
    ElseIf downb = True Then
                    If Not IsNumeric(spool_dist.Text) Then
                        spool_dist.Text = "127"
                    End If
                    Call SendCommand(0, CByte(spool_dist.Text), 0, 6)
                    info.Caption = "down"
    Else
                    info.Caption = "none"
    End If
End Sub

Sub GetJoystick()
    Dim FVel As Double, TVel As Double, RVel As Double, LVel As Double
    Dim JoyInformation As JoyInfo
    Dim RightForward As Boolean, LeftForward As Boolean

    joyGetPos joystick1, JoyInformation

    TVel = JoyInformation.wXpos - CLng(joy_x_center.Text)
    FVel = JoyInformation.wYpos - CLng(joy_y_center.Text)

    If FVel > 0 Then
        FVel = FVel / (CLng(joy_y_max.Text) - CLng(joy_y_center.Text))
    Else
```

```
    FVel = FVel / (CLng(joy_y_center.Text) - CLng(joy_y_min.Text))
End If

If TVel > 0 Then
    TVel = TVel / (CLng(joy_x_max.Text) - CLng(joy_x_center.Text))
Else
    TVel = TVel / (CLng(joy_x_center.Text) - CLng(joy_x_min.Text))
End If

joy_x.Text = TVel
joy_y.Text = FVel

RVel = (FVel + TVel) * 255      'use waiting here
LVel = (FVel - TVel) * 255

LeftForward = False
If (LVel < 0) Then
    LeftForward = True
End If
LVel = Abs(LVel)

RightForward = False
If (RVel < 0) Then
    RightForward = True
End If
RVel = Abs(RVel)

If (LVel > 255) Then
    LVel = 255
End If

If (RVel > 255) Then
    RVel = 255
End If

'send robot new speed to operate at
Call SendCommand(6, CByte(RVel), CByte(LVel), 0)

If LVel + RVel > 1 Then
    'send robot new direction to run at
    If LeftForward And RightForward Then
        info.Caption = "forward"
        Call SendCommand(2, 80, 2, 0)
    ElseIf LeftForward And Not RightForward Then
        info.Caption = "right"
        Call SendCommand(5, 80, 2, 0)
   ElseIf Not LeftForward And RightForward Then
        info.Caption = "left"
        Call SendCommand(4, 80, 2, 0)
    Else
        info.Caption = "reverse"
        Call SendCommand(3, 80, 2, 0)
   End If
Else
    Call SendCommand(1, 0, 0, 0)
    info.Caption = "stop"
```

```
    End If

    joy_rvel.Text = RVel
    joy_lvel.Text = LVel

End Sub

Sub RunApp()
    Dim command As Integer

    quitflag = False

    Do Until quitflag = True
        runinfo.Caption = "runnning"

        If MSComm1.OutBufferCount < 3 Then      'do not overwelm the
serial output
            If input_type = USE_JOYSTICK Then
                Call GetJoystick
            Else
                Call GetButtons
            End If
        End If
        DoEvents
    Loop

    info.Caption = "stopped"
    runinfo.Caption = "stopped"
End Sub

Private Sub back_MouseDown(button As Integer, Shift As Integer, X As
Single, Y As Single)
    b = True
End Sub

Private Sub back_MouseUp(button As Integer, Shift As Integer, X As
Single, Y As Single)
    b = False
End Sub

Private Sub Command1_Click()

End Sub

Private Sub down_MouseDown(button As Integer, Shift As Integer, X As
Single, Y As Single)
    downb = True
End Sub

Private Sub down_MouseUp(button As Integer, Shift As Integer, X As
Single, Y As Single)
    downb = False
End Sub

Private Sub forward_MouseDown(button As Integer, Shift As Integer, X As
Single, Y As Single)
```

```
    'Call SendCommand(2, 500, 0)
    f = True
End Sub

Private Sub forward_MouseUp(button As Integer, Shift As Integer, X As
Single, Y As Single)
    f = False
End Sub




Private Sub joy_DragDrop(Source As Control, X As Single, Y As Single)

End Sub



Private Sub joy_start_Click()
    Call InitJoystick
End Sub

Private Sub left_MouseDown(button As Integer, Shift As Integer, X As
Single, Y As Single)
    l = True
End Sub

Private Sub left_MouseUp(button As Integer, Shift As Integer, X As
Single, Y As Single)
    l = False
End Sub

Private Sub pstop_Click()
    Call SendCommand(0, 0, 0, 0)
End Sub

Private Sub quit_Click()
    quitflag = True
End Sub


Private Sub right_MouseDown(button As Integer, Shift As Integer, X As
Single, Y As Single)
    r = True
End Sub

Private Sub right_MouseUp(button As Integer, Shift As Integer, X As
Single, Y As Single)
    r = False
End Sub

Private Sub run_Click()
    Call RunApp
End Sub
```

```
Private Sub speed_Click()
    If Not IsNumeric(rf_speed.Text) Then
        rf_speed.Text = "127"
    End If

    If Not IsNumeric(lf_speed.Text) Then
        lf_speed.Text = "127"
    End If

    Call SendCommand(6, CByte(rf_speed.Text), CByte(lf_speed.Text), 0)

End Sub

Private Sub stop_Click()
    MSComm1.OutBufferCount = 0
    Call SendCommand(1, 0, 0, 0)
End Sub

Private Sub Text3_Change()

End Sub

Private Sub up_Click()
    Call SendCommand(0, 0, 0, 4)
End Sub

Private Sub up_MouseDown(button As Integer, Shift As Integer, X As
Single, Y As Single)
    upb = True
End Sub

Private Sub up_MouseUp(button As Integer, Shift As Integer, X As
Single, Y As Single)
    upb = False
End Sub

Private Sub up2mag_Click()
    Call SendCommand(0, 0, 0, 5)
End Sub

Private Sub up2mag_MouseDown(button As Integer, Shift As Integer, X As
Single, Y As Single)
    upup = True
End Sub

Private Sub up2mag_MouseUp(button As Integer, Shift As Integer, X As
Single, Y As Single)
    upup = False
End Sub
```

Conclusion

Here are a few brief words on my experience and recommendations if trying to do this project. I had a really great time working on this project although I wish I had more knowledge about other chips and hardware on the market. I guess one of the biggest problems in working on this project was having the right equipment to do the task at hand. The most important piece of equipment you need when trying to do communication is a digital oscilloscope. Do not try to start a project like this without one. Also make sure you have a good understanding of the chips data sheet and the specialty functions. I spent a lot of time trying to configure the chip properly which made me really frustrated and discouraged at times. Finally if the problem seems to be too hard then it probably is because you do not understand something or you made it that way. And always remember to start out small and learn how to use the hardware before trying to do cool things with it.