# A Maze Vacuum Robot
(RoboVac)

Final Report
EE 198 B

Maung Maung
Thomas M Ho

Adviser: Dr. Kwok

12 / 1 / 2004

San Jose Sate University

## Contents

## 1. Abstract

Robotics is a valuable educational tool that introduces students to a variety of different engineering disciplines such as electrical, computer, and mechanical engineering. In order to prove that this work can be accomplished in one semester an autonomous vacuum robot was designed and built. This report describes the development of a vacuum cleaner robot prototype, both on software and hardware point of view. We used 4”x7” circuit board for the body of our robot, and the total size of the robot is about 6 inches in width and 10 inches in length. The height of our robot is about 5 inches. On the board, we build our circuit. Mini Radio Shack wireless vacuum is attached in front of the board and we put the sonar sensor on the top of the mini Radio Sharp vacuum that we used to detect the obstacles when the robot is moving. Both the hardware and software are successfully running simple algorithms for obstacles avoidance and desire destination. This prototype also provides a valuable model for future robot.

## 2. Introduction

The research and development of a robot prototype able to vacuum cleaning a room, an office or even an entire house is not a trivial challenge. In order to tackle such a task, we did our research most of the time since we decided to design the prototype for vacuum robot and we tried to finish our project within one semester with our limited knowledge. Although nowadays, there are different brands of autonomous vacuum robot in the market, the reason that we want to do vacuum robot for our senior project is we want to design our robot as simple as we can and we aware that we live in a world with shortened product life, an increased variety of products and a demand for higher quality on the products. Continuous productivity improvement goes without saying.

At the same time, increased safety reduced environmental impact and better working conditions become more and more important. Not only that we also want to reduce the manufacturing cost. So, we decided to design the prototype for the vacuum robot. The name of our robot is *RoboVac*. *RoboVac* stands for Robotic Vacuum. Our *RoboVac* is a programmable robot. Users can program the robot to go in any path, as they desire. Our robot is not only the prototype for autonomous vacuum but also we can implement to mow robot, street cleaning robot, etc.

## 3. General Design of our Robot

*RoboVac* has four main components. The first component is a sonar sensor. The sensor provides the robot with information about its surroundings. The robot interprets data returned by the sensor about the environment and is able to make decisions accordingly. The second major component of the robot is the micro controller and the memory. The micro controller controls all of the operations of the robot, from collecting sensor data to carrying out appropriate actions. The memory holds the operating system and sensor data.  The third main component of the robot is the program. The program describes a set of rules for the robot to follow. This tells the micro controller how often to sample the sensors, what to do based on the current state of the sensors, and how to implement actions through the output devices. The final component of the robot is the output. This component consists of two stepper motors. The robot interacts with its surroundings using these devices.

The micro controller (PIC16F84) and the memory (PIC16F84) compose the computational hardware of the system. The micro controller coordinates all of the robot's operations; it collects data from the sensors, performs analog-to-digital (A\D) conversions, runs a multi-tasking operating system, runs the consumption program, and controls the stepper motors. PIC16F84 is a widely used micro controller that is flexible enough for a variety of applications. PIC16F84 is the logical choice for these particular applications with its 2 bytes of program memory, 128 bytes in the data EEPROM, 224 bytes data memory. So we use PIC16F84 to store the operating system, the program, and all pertinent information that the robot needs to function.

## 4. How *RoboVac* works?

*RoboVac* has one two-ways toggle switch and two push button switches. Two-ways toggle switch is one side for programmable mode and the other side is for ready to move mode. In order to know clearly which side of toggle switch is what for, we put two different lights in our circuit. If the toggle switch is on the programmable mode, the red light will be on and if the toggle switch is on the ready mode, the green light will be on. For the push button switches, one switch is for the run mode of the robot and the other one is for the reset switch.

### A. Micro controller chip

In our circuit of the robot, we use three 8-bit CMOS micro controllers, which are PIC16F84A. We use one PIC16F84A for PC-EEPROM Interface and the other two are for Main Control and Motor Control of the robot.

First of all, users have to know how to write a program to PC-EEPROM Interface PIC. If users want to write a program to PC-EEPROM Interface PIC, he/she have to put the toggle switch to program mode. *RoboVac* is a programmable robot. Programmable means user can program it to follow a certain path as defined by 6 instructions that we have developed. Using as many combinations of these six instructions as necessary, users can program the robot to go in any path they desire. They are as follow.

- S = Stop
- F = Forward
- B = Backward

- L = Left
- R = Right
- D = Done

The above 6 instructions work by associating a value to it. They are written as a text file. After that, using the GUI interface application (which is used to communicate with the robot when programming the above 6 instructions), these commands are programmed into the robot's on board EEPROM (memory) and then executed by the robot's control circuit.

For example, if the user wants the robot to go

Forward 10 inches

Right turn of 45 degrees

Stop for 2 seconds

Done.

The following instructions will write in the EEPROM.

- F 10
- R 45
- S 2
- D 0

We have to note that value 0 must be provided after D because the above instructions work as (command, value) pair.

## B. Main-Control and Sonar sensor

After the PCEEPROM chip read the instructions that the users write, PCEEPROM PIC gives these instructions to the Main-Control PIC. Main-Control PIC will command the robot to move according to the users instructions and in the mean time sonar sensor (SFR04) will check the obstacles in front of the robot. If the sonar sensor finds the obstacles in front of the robot, the robot will stop **one inch** before the obstacles and then looks for the left side of the robot clear or not. If the left side is clear, the sonar sensor will give command to the Main-Control PIC to turn right. After the robot turns left, the sensor will start collecting the length from the robot starting making turn left point to the point obstacles clear. After the obstacles clear, the robot will move about the length of the robot according to the program that we write in the Main-Control PIC and then make right turn and go straight. When the robot goes straight, the sensor will check the right side for the obstacles clear or not. When the obstacles clear, the robot will turn right and go back to the same length that the sensor collects first. After that the robot will turn left and follow the rest of the instructions that the users write in the PCEEPROM.

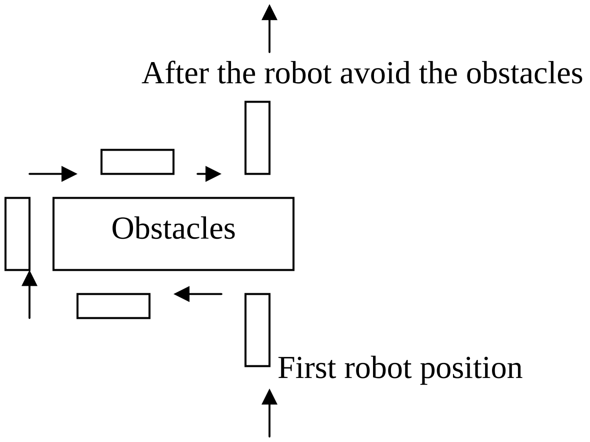

Fig 1: The movement of the robot before and after the obstacles

But, if the obstacle is on the left side of the robot, the sensor will turn to the right and check the obstacle again. If there is no obstacle on the left, the robot will turn to the left and follow the same instruction as right turn.

**C. Motor**

*RoboVac* has two unipolar stepper motor drive IC (UCN5804B) underneath the wire-wrapped board. One stepper motor drive IC is for the right turn and the other one is for the left turn. The reason why we use the stepper motor for our robot is stepper motors behave differently than the standard DC motors. A stepper motor can be a good choice whenever controlled movement is required. They can be used to advantage in applications where we need to control rotation angle, speed, position and synchronism.

## 5. Things that we learnt from our project

We built two different robots for our project.

### A. First Robot

When we built our first robot, we had a lot of difficulties. For example, wire wrapping. The first time that we built our circuit on the wire-wrapped board, we did not do neatly and we found our later that some of the wires are shorted. So, we need to rebuild for the second time. From that we got the experience with the wire wrapping. After we finished building our first robot, we started testing our first robot. Our first robot did not follow the instructions that the users wrote on the PCEEPROM PIC. For example, if the users want to make a right turn 90 degrees, the robot will not turn 90 degrees. Robot will turn whatever direction and after that all the rest of the instructions will be wrong.  At that time, we did not know what was the problem.

# Our First Robot’s Design

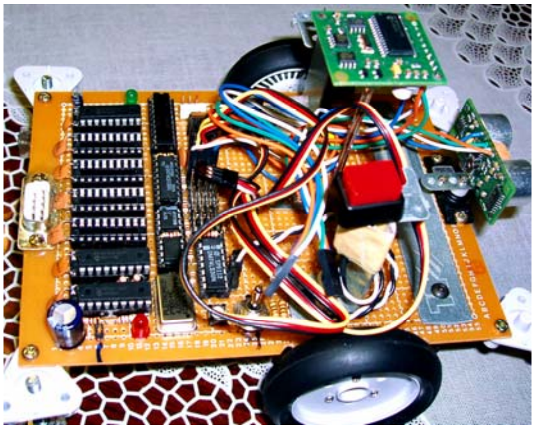

Fig 2: Top view of the first robot

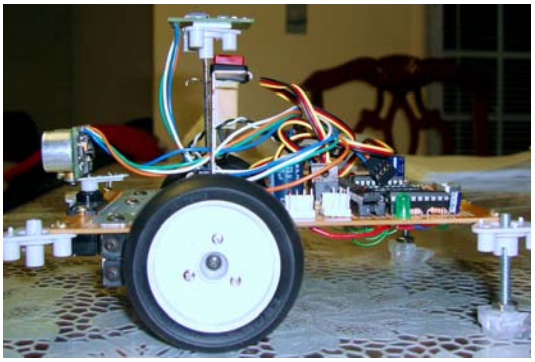

Fig 3: Side view of the first robot

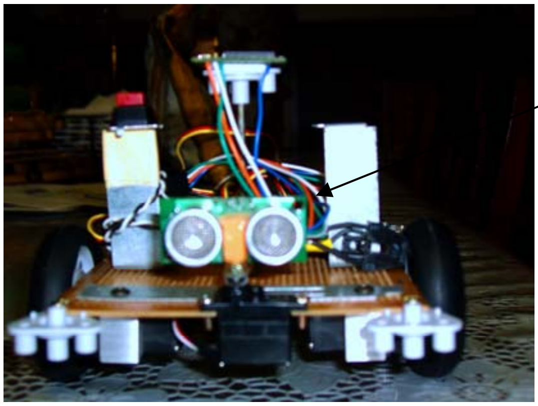

Fig 4: Front view of the first robot

### B. Second Robot (*RoboVac*)

We did not know what was the problem, so we build the new robot for the second time with the new wire-wrapped board. For the second robot, we used the same parts as the first robot except the motor. Although we used the servo motor for the first robot, for the *RoboVac*, we used the stepper motor because when we did the research for the motor, we found out that stepper motor are more accurate and precise when the wheel makes a turn.

After we finished building our *RoboVac*, we understood why the first robot did not turn accurately. Stepper motor can be control rotation angle, speed and position. After we solved the problem of the hardware of the robot, we had another problem coming up. When we tested our robot, the robot did not avoid the obstacle all the time. Our robot just went all the way and hit the obstacles even though the robot detected the obstacles. So we checked our program codes with step-by-step movement of the robot. We found out that we needed to put stop for a while command in our software program. For example, the robot is going forward and when the sonar sensor is detect the obstacle about one inch, the robot will stop for a while (depend on the time that we write in our software program) and the sensor will check the left side for rotate.

# The second Robot (*RoboVac*)'s design

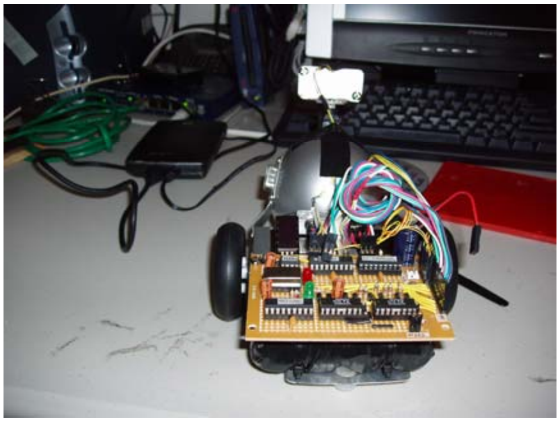

Fig 5: RoboVac's circuit picture

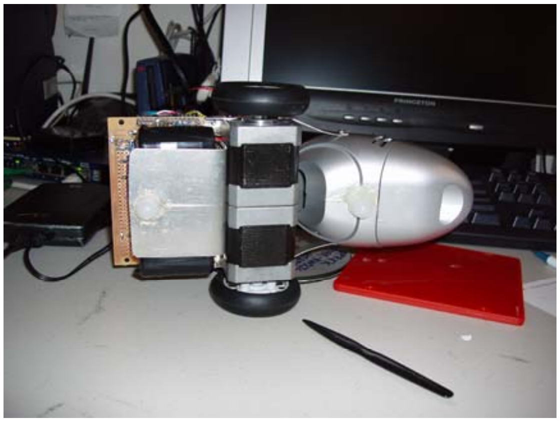

Fig 6: Bottom view of the robot

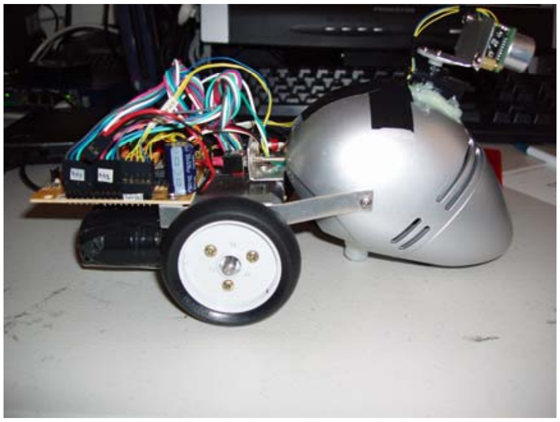

Fig 7: Side view of the robot

## 6. Materials List (Part and Component in Details) for the *RoboVac*

| | Quantity |
|---|---|
| Small wire-wrapped board (for the circuit) | 3 |
| Push button switch (for reset and run) | 2 |
| 2-way Toggle switch (for ready/ program mode) | 1 |
| Run switch 2 pin header | 1 |
| Ready/ Program switch 3 pin header | 1 |
| DB-9 Male connector | 1 |
| DB-5 Male connector | 1 |
| 5 pin header (for M1 and M2 drivers) | 2 |
| Sonar Sensor (SFR04) | 1 |
| Oscillator 4MHz | 1 |
| 8 bit CMOS micro controller (PIC 16F628) (for PC-EEPROM Interface, Main Control and Motor Control) | 3 |
| Unipolar stepper motor drive IC (UCN5804B) | 1 |
| Stepper Motor | 2 |
| Wheels | 2 |
| Mini Mouse Vacuum (from Radio Shack) | 1 |

## 7. Cost Analysis for RoboVac

| | |
|---|---|
| Motor (DC, Servo, Step) | $100 |
| Micro controller chip | $50 |
| Sonar | $35 |
| 3 Vacuums | $150 |
| Circuit board | $20 |
| Push bottom switch | $10 |
| 2 way toggle switch | $3 |
| D-B 9 male | $7 |
| Oscillator | $5 |
| Wheel | $8 |
| Component | $100 |
| Total cost | $488 |

## Time Schedules for our project

| | |
|---|---|
| June | Finished researching for our robot |
| July | Ordered the materials that we needed for our robot |
| August | Build our first robot |
| September | Troubleshot and tested our first robot |
| October | Redesigned the new robot |
| November | Tested, troubleshot and finished our *RoboVac*<br>Prepared the report for our project. |
| December | Prepared for the presentation of our project |

.

# 9. Circuit Diagrams for our Robot

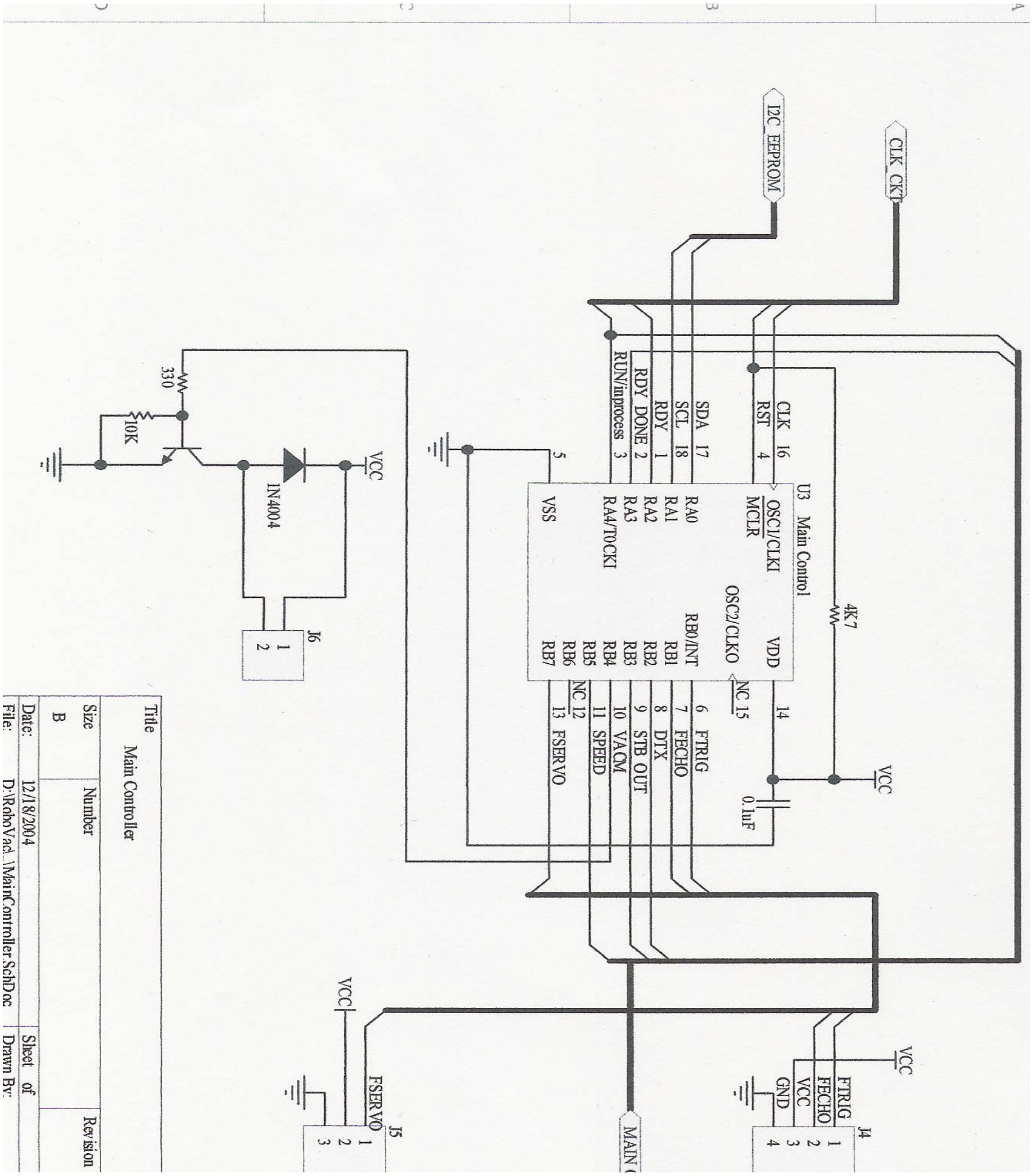

Fig.5. Main controller circuit

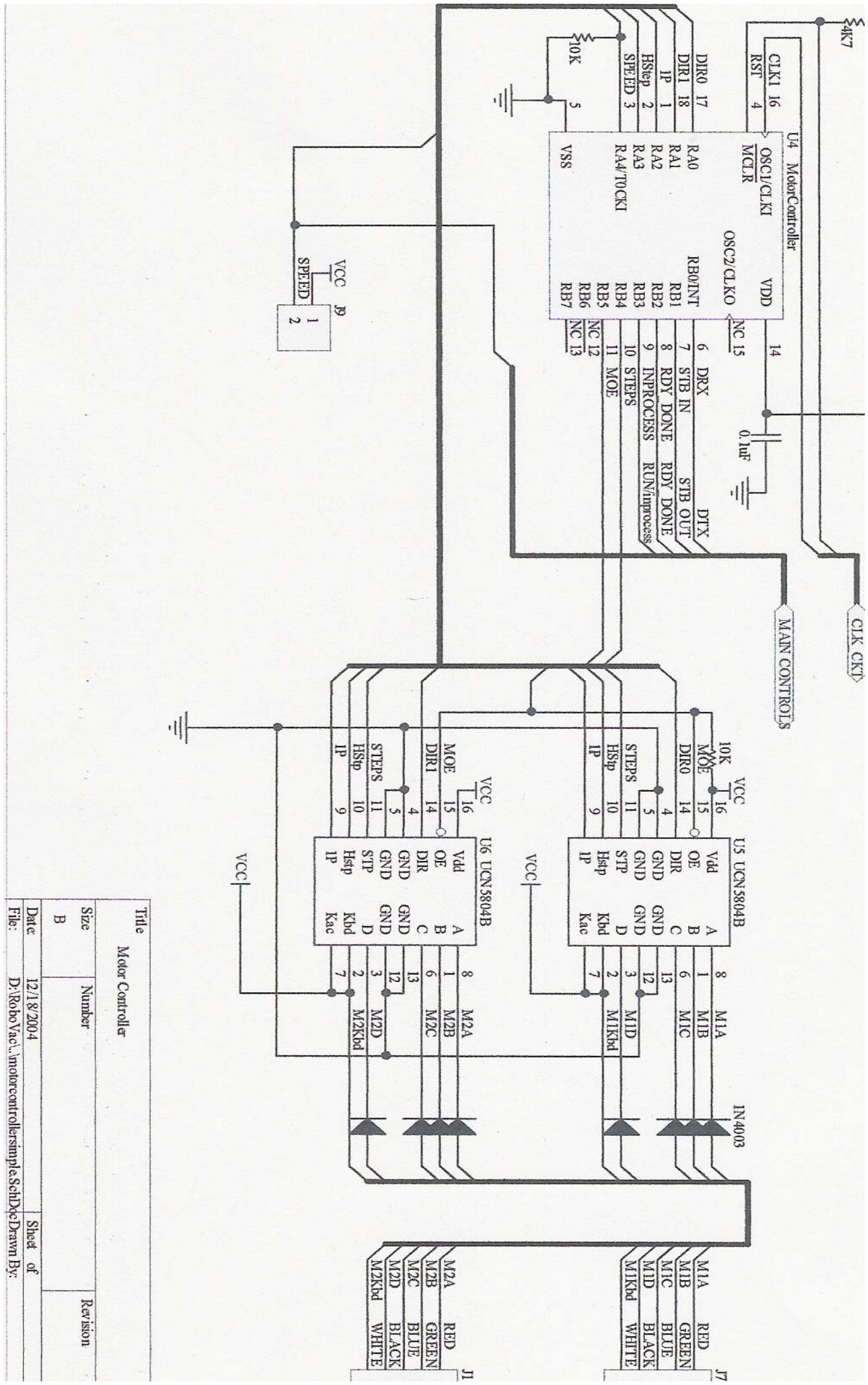

Fig 7. Motor controller circuit

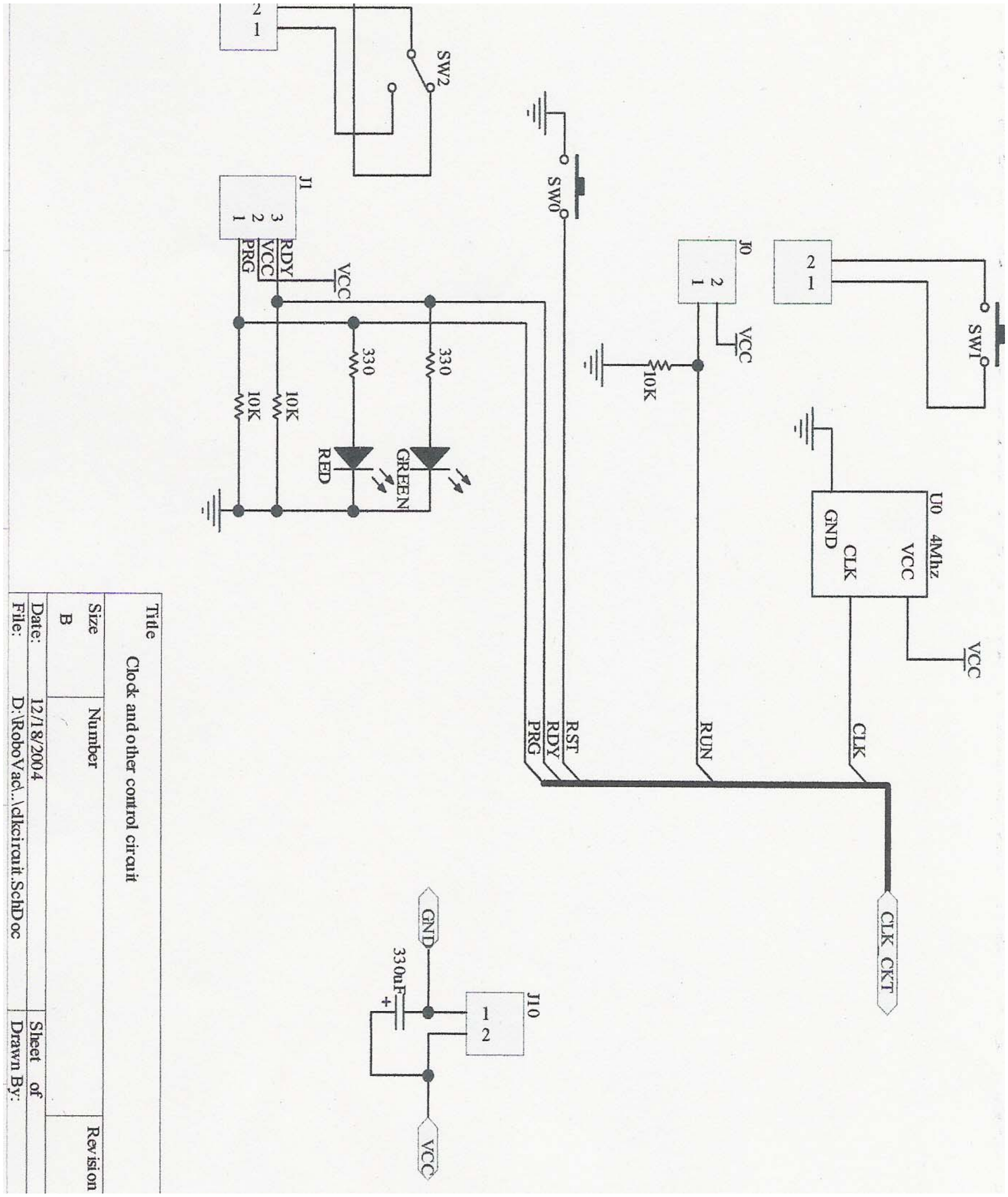

Fig 9. Clock and other control circuit

## Design Units ID

| | |
|---|---|
| J0 | Run sw 2-pin header |
| J1 | Rdy/Prg sw 3-pin header |
| J2 | DB-9 Male Connector |
| J3 | DB-9 5-pin header |
| J4 | 4-pin Front Sonar Header |
| J5 | 3-pin Front Sonar Servo Header |
| J6 | 2-pin Vacuum Motor |
| J7 | 5-pin M1 Header |
| J8 | 5-pin M2 Header |
| J9 | 2-pin Speed Selection Jumper |
| J10 | 2-pin Header for Power |
| | |
| U0 | Oscillator 4Mhz |
| U1 | PC-EEPROM Interface PIC (16F84A) |
| U2 | 24C16 I2C EEPROM (2k x 8) |
| U3 | Main Control PIC (16F84A) |
| U4 | Motor Controller PIC (16F84A) |
| U5 | M1 Driver (UCN5804B) |
| U6 | M2 Driver (UCN5804B) |
| | |
| SW0 | Push button reset sw |
| SW1 | Push button run sw |
| SW2 | 2-way Rdy/Prg toggle sw |

## 10. Program codes for *RoboVac*

See appendix

## 11. Conclusion

The final design accomplished all of the goals initially set. The robot performs better than had been anticipated. The design chosen is not only very flexible, but also sturdy. New features can be easily added to the robot. Completion of this project brings a new product to the world of industry to increase speed and efficiency while reducing the loss. In developing this project, new and innovative solution were needed to tackle the design challenges that were encountered. Each problem was dealt with further research and trial and error method in a timely manner. Overall the learning objective of this project provided an opportunity to research beyond the academic requirements.

# 12. Appendix:

## A. Main controller

```
'filename: maincontroller84A.pbc
'purpose: processes the CW by reading from the eeprom, sending it to the appropriate
'device (for now, there is only one device: the motor controller)


'Variables declarations
Symbol portadata   = B0  'porta data is read into this variable
Symbol control     = B2  'control byte for i2c command
Symbol addr        = B3  'address byte for the i2c eeprom
Symbol totalcmd    = w2  'total read/write from/to pc or eeprom (B4,B5)
Symbol data        = W3  'working data variable (B6,B7).
Symbol cmd         = B8  'PC command R/W/T etc... during programming
Symbol turns       = B9  '
Symbol fservopos   = B10 'front sonar servo position
Symbol totalpulse  = B11 'the number of pulses required to turn the servos
Symbol err         = B12 'indicates if the path is blocked (++)
Symbol originalcmd = B13 'to save the original command when there is error
Symbol mindist     = B14
Symbol fmindist    = B15
Symbol seq5delay   = W9
Symbol fdist       = W10 'holds the front sonar's distance (B16,B17)
Symbol initfdist   = W11 '(B22,B23)
Symbol trisa       = $85 'porta direction reg
Symbol porta       = $5  'porta data reg
Symbol servodelay  = 7   '7ms servo delay
Symbol rightpos    = 253 '2550 us for right servo position
Symbol centerpos   = 148 '1480 us for the servo center position
Symbol leftpos     = 53  '540 us for left servo position
Symbol lrmindist   = 150 'left and right minimun distance for turning
'pin assignments and other declarations
Symbol FTRIG       = 0   'front sonar trigger output pin
Symbol FECHO       = 1   'front sonar echo input pin
Symbol DTX_DRX     = 2   'In-Circuit serial data I/O pin
Symbol STB         = 3   'In-Circuit data out strobe pin
Symbol VACM        = 4   'vacauum motor control pin
Symbol SPEED       = pin5'speed input pin
Symbol FSERVO      = 7   'Front sonar servo ouput pin
Symbol S           = 0   'value for Stop
Symbol F           = 1   '         Forward
Symbol B           = 2   '         Backward
Symbol L           = 3   '         Left
Symbol R        = 4   '        Right
Symbol D        = 5   '        Done
Symbol off      = 0   '        low
Symbol on       = 1   '        high


'--------------------------------------------------------------------------------
'main program flow (corrosponds to maincontrollerflowchart.sdr)
'--------------------------------------------------------------------------------

'initialize the variables and center the front servo
Initialize:
Poke trisa,%00011111 'make porta[4:0] input
Low FTRIG          'disable sonar trigger pin
Low STB           'disable the data strobe pin
Low DTX_DRX        'keep the serial data I/O pin low
Low FSERVO         'disable the front servo
Low VACM           'turn off the vaccumm motor
fservopos = centerpos'center the servo
totalpulse = 100     '100 pulses initially
GoSub RotateServos   'center the servos initially
totalpulse = 50      'default is 60
fmindist = 25
seq5delay = 250
IF SPEED = off Then Main
fmindist = 40
seq5delay = 400


'Main Loop. keep checking the ready switch(read through a.3)
'keeps the robot idle if it's not on.
'prepare to run if it's on.
Main:
Peek porta,portadata       'read porta
IF bit2 = off Then Main    'if ready switch is off, keep checking


'Ready Mode. RDY switch is on. continue checking RUN and RDY switch.
'RUN is read through a.4, RDY is read through a.2
'if RDY and not RUN, repeat.
'if not RDY then go back to main.
RdyMode:
Peek porta,portadata       'read porta
IF bit2 = off Then Main    'if ready switch is off, go back to main
IF bit4 = off Then RdyMode 'if run switch is off, keep looping
Pause 500                  'pause 1/2 s after pressing the run switch
```

```
'-------------------------------------------------------------------------------------------------
'RunMode program flow
'-------------------------------------------------------------------------------------------------

'the ready switch was on and the run switch was pressed.
'execute each CW by reading from the EEPROM and sending it to the motor
'controller one by one.
'right after sending the CW to the motor controller, check the front path
'and react as appropriate if blocked.
RunMode:
control = %01010111      'go to bank7
addr = %11111110         'to the last two bytes(the totalcmd value)
GetTotalCmd:
GoSub GetEEpromData      'read the total number of commands (will be in data (B6,B7)
)
totalcmd = data          'assign the data to totalcmd variable
control = %01010000      'go back to bank0
addr = 0                 'to read from the first byte
GetCW:
IF totalcmd <= 0 Then Initialize 'if there are no commands, start over from main
GoSub GetEEpromData      'get the CW
cmd = B6 & %00000111     'get the command part
IF cmd = D Then Initialize 'if command is DONE, then nothing to do.
GoSub ExecuteCmd         'execute the command
Low VACM                 'turn off the vaccumm motor
IF addr = 0 Then SwitchBank'if 0-255-0 rolls over, time to switch bank
DoneSB:
totalcmd = totalcmd - 1  'decrement total command
GoTo GetCW               'Repeat
SwitchBank:
control = control + 1    'switch to the next bank (3 LSb of control)
GoTo DoneSB              'done switching bank


'At this point, CW has been read from the memory.
'It is sent to the motor controller to be executed.
ExecuteCmd:
GoSub SendCW             'first, send the CW
IF cmd = S Then Stop     'if current cmd is stop, then just stop
IF cmd = F Then CheckFPath 'if forward, then must check the front path
Turning:                 'if L or R
Peek porta,portadata     'check the inprocess signal
IF bit4 = on Then Turning  'wait while inprocess signal is high (still turning)
GoTo CmdExecuted         'done
CheckFPath:              'Check the path in the front
```

```
High VACM                 'turn the vaccumm motor on
GoSub CheckPath           'check for things blocking the current path
IF err = on Then FixError  'if current path is blocked, Fix it
ErrorFixed:
Peek porta,portadata        'path clear. now check the inprocess signal
IF bit4 = on Then CheckFPath 'Keep driving if the cmd is still inprocess
CmdExecuted:
Return                    'execution completed


'check the path of the direction set by cmd.
'if an object is at the mindist, err = on,indicating path is blocked.
'else, err = off
CheckPath:
err = off              'off initially
IF cmd = F Then CheckF 'Check the corrosponding directions
IF cmd = L Then CheckL
IF cmd = R Then CheckR
GoTo PathChecked       'if none of the above cmd, then simply bail
CheckF:                'cmd was F so check the front path
mindist = fmindist          'set minimun distance to 25 sonar unit (6 unit = 1cm)
fservopos = centerpos  'position the servo to the center (point forward)
GoTo AdjustServo       'adjust the servo
CheckR:                'cmd was R so check right
mindist = lrmindist    'set up the distance
fservopos = rightpos   'turn the servo to that direction
GoTo AdjustServo
CheckL:
mindist = lrmindist
fservopos = leftpos
AdjustServo:
GoSub RotateServos
GoSub GetFDist         'get the distance
IF fdist > mindist Then PathChecked 'if cleared
err = on               'not cleared so err = 1
PathChecked:
Return

FixError:
originalcmd = cmd          'first save the original cmd
data = S                  'stop the motors
GoSub SendCW
TryR:                     'check R first
cmd = R
GoSub CheckPath
IF err = off Then StartSeq1 'R is clear
```

```
cmd = L                 'R is blocked so check L
GoSub CheckPath
IF err = off Then StartSeq1 'L is clear
Goto LRBlocked              'both R and L are blocked. at this point, just end
SeqDone:                 'The fix sequences return here when done
cmd = originalcmd           'restore the original cmd
GoTo ErrorFixed            'and go back

LRBlocked:
end


StartSeq1:               'The start of the fix sequence
data = cmd                'set data to current cmd since it's used to send to motor controller
GoSub ExecuteCmd            'The cmd has been set by the calling subroutine.

'-------------------------  testpoint 1

IF cmd = L Then RSequence   'turn the servos R if cmd is L
LSequence:                'cmd was R so turn the servos to the L
fservopos = leftpos         'set the servo positioin value
GoTo TurnServos             'turn the servo
RSequence:                'cmd was L so start R sequence
fservopos = rightpos        '
TurnServos:
GoSub RotateServos


GetInitFDist:              'first get the initial side distance
GoSub GetFDist
initfdist = fdist + 25                'pad 25 extra distance
IF initfdist >= lrmindist Then SkipSeq2  'if there is no object on the side

'------------------------ testpoint 1

Seq2:                    'keep checking the side until it becomes cleared
GoSub GetFDist
IF fdist <= initfdist Then Seq2

SkipSeq2:
High STB

'end
'------------------------  testpoint 2

Seq3:                   'wait until turned
```

```
Peek porta,portadata         'read porta data to check for the inprocess signal
IF bit4 = on Then Seq3
Low STB

'end
'-----------------------  testpoint 3 and 4

GoSub GetStepsTraveled       'at this point, ready to go F
initfdist = initfdist - 20   'first get the steps traveled
Seq4:
Peek porta,portadata
IF bit4 = off Then SeqDone   'if original distance is reached, done
GoSub GetFDist
IF fdist < initfdist Then ToSeq5  'keep going until the first contact with the object
GoTo Seq4


ToSeq5:                      'the object is just detected.
initfdist = initfdist + 75
Pause seq5delay
High STB
Pause 25
Low STB

'end
'----------------------- testpoint 5

Seq5:                        'next go F until original distance is reached or
Peek porta,portadata         'the object just becomes cleared
IF bit4 = off Then SeqDone
GoSub GetFDist
IF fdist < initfdist Then Seq5
High STB

'end
'----------------------  testpoint 6, 7,8

FinalSequence:               'at this point, F is done.
Peek porta,portadata           'read porta data to check for the RDY_DONE signal
IF bit3 = off Then FinalSequence
Low STB

'---------------------- testpoint 9

GoTo SeqDone
```

```
'---------------------------------------------------------------------------------------------
'Helper subroutines
'---------------------------------------------------------------------------------------------


'Stop the robot
Stop:
data = data / 8
data = data * 1000
Pause data
GoTo CmdExecuted

'send the CW to the motor controller
SendCW:
High STB                'assert the data strobe line
WaitForRDY:             'wait for rdy acknowledgement from other PIC
Peek porta,portadata    'read porta data to check for the RDY_DONE signal
IF bit3 = off Then WaitForRDY
SendData:
SerOut DTX_DRX,N9600,(B6,B7)   'send the data
WaitForDone:            'wait until data was received successfully
Peek porta,portadata    'read porta data to check for the RDY_DONE signal
IF bit3 = on Then WaitForDONE
Low STB                 'data received successfully, disable the STB line
Return

'get the steps traveled from the motor controller
GetStepsTraveled:
SerIn DTX_DRX, N9600, B22, B23
initfdist = initfdist * 10
initfdist = initfdist / 36
Return

'turn the servos to fservopos
RotateServos:
For turns = 0 to totalpulse
PulsOut FSERVO,fservopos
Pause servodelay
Next turns
Return

'get the distance from the sonar
GetFDist:
PulsOut FTRIG,1         'send 10us pulse to activate the front sonar
PulsIn FECHO,1,fdist   'measure the high pulse and place width*10 in fdist
Pause 8
```

```
Return                 'done

'get the CW from eeprom
GetEEpromData:                  'NOTE: data read is stored in (B6,B7).
I2CIN control,addr,B6           'first read the low byte
addr = addr + 1                 'increment to the next byte
I2CIN control,addr,B7           'read the high byte
addr = addr + 1                 'increment
Return                          'done reading so return

'------------------------------------------------------------------------------------------
```

## B. Motor controller

```
'filename: motorcontroller.pbc
'purpose: execute the commands and drive the motors accordingly


Symbol trisa        = $85   'porta direction register
Symbol porta        = $5    'porta data register
Symbol portadata    = B0    'holds the data read from porta
Symbol cmd          = B1    'holds the command
Symbol totalsteps   = W1    'the total number of steps required to go(B2,B3)
Symbol cursteps     = W2    'keeps track of current number of steps(B4,B5)
Symbol fsteps1      = W3    'keeps track of the total F steps during the fix sequence
Symbol offsetsteps  = W4    'the offset distance
Symbol stepsleft    = W5
Symbol delay        = B12   'delay between each step
Symbol on           = 1     'high
Symbol off          = 0     'low
Symbol S            = 0     'value for Stop
Symbol F            = 1     '         Forward
Symbol B            = 2     '         Backward
Symbol L            = 3     '         Left
Symbol R            = 4     '         Right
Symbol D            = 5     '         Done
Symbol FDIR         = 9     'forward direction %01001
Symbol BDIR         = 10
Symbol RDIR         = 8
Symbol LDIR         = 11
Symbol NINETYD      = 214   'the number of steps required for 90 deg turn
Symbol seqdelay     = 150   'the delay between each sequence

'pin assignments
Symbol DRX_DTX   = 0     'serial data input pin from the main PIC
Symbol STB       = pin1  'data strobe input from the miain PIC (++)
Symbol RDY_DONE  = 2     'handshake pin RDY is (++), DONE is (--)
Symbol INPROCESS = 3     'indicates while a cmd is being executed
Symbol STEPS     = 4     'step pulse output pin
Symbol MOE       = 5     'Motor Output Enable pin


Initialize:
Poke trisa,%00010000   'make porta pins outputs
Peek porta,portadata
delay = 8
IF bit4 = off Then LowSpeed
delay = 4
```

```
LowSpeed:
Low RDY_DONE          'just disable these pins
Low INPROCESS         '
Low STEPS             'stop the motor
High MOE              'disable the motor outputs
offsetsteps = 335     '


'check the STB signal for data transfer.
Main:
IF STB = off Then Main 'if data is about to be transfered


'execute the commands
RunMode:
GoSub GetCW
High INPROCESS              'cmd is being executed so set it H
cursteps = 0                'reset it
Branch cmd,(CmdExecuted,Forward,Backward,Left,Right,CmdExecuted)
CmdExecuted:
GoTo Initialize


Forward:
Poke porta,FDIR          'dir0 = 1, dir1 = 0 to go forward
GoTo Drive
Backward:
Poke porta,BDIR          'dir0 = 0,dir1 = 1 to go backward
GoTo Drive
Right:
Poke porta,RDIR          'dir0=dir1=1 to go right
GoTo Drive
Left:
Poke porta,LDIR          'dir0=dir1=0 to go left
Drive:
Low MOE                  'enable the motor outputs
RunMotors:
IF cursteps >= totalsteps Then CmdExecuted 'check remaining number of steps
GoSub SendStepPulse       'send the step pulse
IF STB = off Then RunMotors 'if during cmd execution stb is high, then err
FixSeq:
stepsleft = totalsteps - cursteps  'subtract steps taken so far
cursteps = 0

FixWait:                   'wait until the command is L or R
GoSub GetCW                  'get the next CW
```

```
IF cmd = S Then FixWait
IF cmd = L Then StartLSeq   'check cmd and fix accordingly


'the
StartRSeq:            'R fix sequence
Poke porta,RDIR          'set motor direction to R
RSeq1:
GoSub Turn90            'turn R 90
PulsOut INPROCESS,8         'done R 90. send 50us low pulse

'-----------------------------testpoint 1

cursteps = 0          'prepare to go F
Poke porta,FDIR         'set motor direction to F
Pause seqdelay
RSeq2:
GoSub SendStepPulse        'Go F
IF STB = off Then RSeq2     'If the sonar is still blocked, keep going F

Pause seqdelay
'end
'-----------------------------testpoint 2

fsteps1 = cursteps + offsetsteps'sonar just becomes clear. save the F steps so far
cursteps = 0          'reset and prepare to go for the offset
offsetsteps = 335
GoSub Offset           'Go for the offset steps

Pause seqdelay
'end
'-----------------------------testpoint 3

cursteps = 0          'prepare to turn L 90
Poke porta,LDIR         'set motor direction to L
RSeq3:
GoSub Turn90            'Turn L 90
PulsOut INPROCESS,8         'Done L 90. send 50us low pulse

'end
'-----------------------------testpoint 4

Pause seqdelay
GoSub SendStepsTraveled

Poke porta,FDIR         'set motor direction to F
```

```
cursteps = 0                    'prepare to go F
RSeq4:
GoSub SendStepPulse             'Drive the motor
IF cursteps >= stepsleft Then CmdExecuted
IF STB = off Then RSeq4         'keep going F until the sonar is blocked

Pause seqdelay
'end
'------------------------------testpoint 5

ToSeq5:
GoSub SendStepPulse
IF cursteps >= stepsleft Then CmdExecuted
IF STB = off Then ToSeq5      'keep going F until the sonar is clear
Pause seqdelay

'end
'------------------------------

stepsleft = stepsleft - cursteps
cursteps = 0
offsetsteps = 385
GoSub Offset                    'here, the sonar just cleared. go for the offset steps
IF cursteps >= stepsleft Then CmdExecuted
stepsleft = stepsleft - offsetsteps

Pause seqdelay
'end
'------------------------------testpoint 6

cursteps = 0                    'Prepare to Turn L
Poke porta,LDIR                    'set motor direction to L
RSeq5:
GoSub Turn90       'Turn 90 degrees
Pause seqdelay

'------------------------------testpoint 7

cursteps = 0                   'prepare to go F
Poke Porta,FDIR                   'set motor direction to F
RSeq6:
GoSub SendStepPulse
IF cursteps < fsteps1 Then RSeq6
Pause seqdelay

'------------------------------testpoint 8
```

```
Poke porta,RDIR              'prepare to turn R 90
cursteps = 0
RSeq7:
GoSub Turn90                 'L 90
PulsOut RDY_DONE,8           'done with the sequence. send a 5us high pulse
Pause seqdelay

'------------------------------testpoint 9

cursteps = 0
totalsteps = stepsleft
GoTo Forward                 'go back to forward for now.

'------------------------------Right Sequence completed
'------------------------------------------------------------------------------------------------



StartLSeq:                   'R fix sequence
Poke porta,LDIR              'set motor direction to R
LSeq1:
GoSub Turn90                 'turn R 90
PulsOut INPROCESS,8          'done R 90. send 50us low pulse

'------------------------------testpoint 1

cursteps = 0                 'prepare to go F
Poke porta,FDIR              'set motor direction to F
Pause seqdelay
LSeq2:
GoSub SendStepPulse          'Go F
IF STB = off Then LSeq2      'If the sonar is still blocked, keep going F

Pause seqdelay
'end
'------------------------------testpoint 2

fsteps1 = cursteps + offsetsteps'sonar just becomes clear. save the F steps so far
cursteps = 0                 'reset and prepare to go for the offset
offsetsteps = 335
GoSub Offset                 'Go for the offset steps

Pause seqdelay
'end
'------------------------------testpoint 3
```

```
cursteps = 0             'prepare to turn L 90
Poke porta,RDIR              'set motor direction to L
LSeq3:
GoSub Turn90                'Turn L 90
PulsOut INPROCESS,8              'Done L 90. send 50us low pulse

'end
'------------------------------testpoint 4

Pause seqdelay
GoSub SendStepsTraveled

Poke porta,FDIR              'set motor direction to F
cursteps = 0             'prepare to go F
LSeq4:
GoSub SendStepPulse             'Drive the motor
IF cursteps >= stepsleft Then CmdExecuted
IF STB = off Then LSeq4     'keep going F until the sonar is blocked

Pause seqdelay
'end
'------------------------------testpoint 5

LTestSeq5:
GoSub SendStepPulse
IF cursteps >= stepsleft Then CmdExecuted
IF STB = off Then LTestSeq5    'keep going F until the sonar is clear
Pause seqdelay

'end
'------------------------------

stepsleft = stepsleft - cursteps
cursteps = 0
offsetsteps = 385
GoSub Offset                'here, the sonar just cleared. go for the offset steps
IF cursteps >= stepsleft Then CmdExecuted
stepsleft = stepsleft - offsetsteps

Pause seqdelay
'end
'------------------------------testpoint 6

cursteps = 0             'Prepare to Turn L
Poke porta,RDIR              'set motor direction to L
```

```
LSeq5:
GoSub Turn90       'Turn 90 degrees
Pause seqdelay

'-----------------------------testpoint 7

cursteps = 0                 'prepare to go F
Poke Porta,FDIR              'set motor direction to F
LSeq6:
GoSub SendStepPulse
IF cursteps < fsteps1 Then LSeq6
Pause seqdelay

'-----------------------------testpoint 8

Poke porta,LDIR              'prepare to turn R 90
cursteps = 0
LSeq7:
GoSub Turn90                 'L 90
PulsOut RDY_DONE,8           'done with the sequence. send a 5us high pulse
Pause seqdelay

'-----------------------------testpoint 9

cursteps = 0
totalsteps = stepsleft
GoTo Forward                 'go back to forward for now.

'-----------------------------Right Sequence completed


'----------------------------------------------------------------------------
'helper subroutines
'----------------------------------------------------------------------------

'Get the CW from the main controller
GetCW:
High RDY_DONE                'assert RDY signal(L-H transition) to begin
SerIn DRX_DTX,N9600,B2,B3    'then read the data into B2,B3(totalsteps)
cmd = B2 & %00000111         'extract only the 3LSb (the command part)
totalsteps = totalsteps / 8 'SHR 3 to get the value part
Low RDY_DONE                 'Data is successfully received
Return

'sends the number of steps travelled
SendStepsTraveled:
```

```
SerOut DRX_DTX,N9600,(B6,B7)
Return

'send a 10us pulse to the motor driver chip
SendStepPulse:
PulsOut STEPS,1
cursteps = cursteps + 1
Pause delay
Return

'go F offset distance
Offset:
GoSub SendStepPulse
IF cursteps < offsetsteps Then Offset
Return

'turn approxmately 90 degrees
Turn90:
GoSub SendStepPulse
IF cursteps < NINETYD Then Turn90
Return
```

## C. PC-EEPROM interface84A

```
'filename: pc-eeprominterface84A.pbc
'purpose: interfaces to the PC terminal program for eeprom programming.
'IMPORTANT PRECAUTION: this program is designed to run on 16F84A at 4Mhz
only.

'Programming the hex file.
'the following configuration bits must be set when programming the hex code.
'oscillator = XT
'code-protection = OFF


'Variables declarations
Symbol portadata   = B0  'porta data is read into this variable
Symbol control     = B2  'control byte for i2c command
Symbol addr        = B3  'address byte for the i2c eeprom
Symbol totalcmd    = w2  'total read/write from/to pc or eeprom (B4,B5)
Symbol data        = W3  'working data variable (B6,B7).
Symbol cmd         = B8  'PC command R/W/T etc... during programming
Symbol trisa       = $85 'porta direction reg
Symbol porta       = $5  'porta data reg

'pin assignments and other declarations
Symbol DTX_J2RX    = 0    'serial data in pin from PC
Symbol DRX_J2TX    = 1    'serial data out pin to PC
Symbol J2TX_STB    = pin2 'Serial port data strobe in pin
Symbol on  = 1
Symbol off = 0
Symbol TWR = 3          'eeprom write time

'-------------------------------------------------------------------------------
'main program flow (corrosponds to maincontrollerflowchart.sdr)
'-------------------------------------------------------------------------------

'initialize the variables
Initialize:
Poke trisa,%00000111 'make porta [2:0] input

'Main Loop. keep checking the program switch (read through porta).
'go to the programming mode if it's on.
Main:
Peek porta,portadata       'read porta and check the corrosponding bits.
IF bit2 = off Then Main    'if program switch is off keep looping

'Programming mode.
```

```
'first, check J2TX_STB if the PC is trying to send a data. if not, then check if
'the program switch is still on (ie. the user could decide to cancle the programming).
'if the program switch is off, then go back to main. if it's still on and
'if J2TX_STB = 1 follow by a 0 (a 31ms min pulse), then get the data
'(the initial command) as follow.
'R initiates eeprom read.
'W initiates eeprom write.
'T initiates test (the robot's movements).
'if none was received, start over from main.
Program:
Peek porta,portadata             'read porta and check the ready switch.
IF bit2 = off Then Main          'if the program switch is turned off, go back to main.
IF J2TX_STB = off Then Program   'check if the PC is about to send initial cmd.
Pause 25                         'at 16Mhz, PIC16F628 executes at 4 times the speed. so 100 =
25ms.
IF J2TX_STB = off Then Program   'if the pulse width is less than 25ms
GetInitCmd:
GoSub GetPCData                  'first, get the initial command (R/W/T etc...)
cmd = B6                         'get the command part
IF cmd = "R" Then ReadEEprom     'R = PC to read from eeprom
IF cmd = "W" Then WriteEEprom    'W = PC to write to eeprom
GoTo Initialize                  'start over from main if any other charcter was received.


'-----------------------------------------------------------------------------------------------
'eeprom read program flow
'-----------------------------------------------------------------------------------------------

'Reading from the external i2c EEPROM.
'first read the value of the total number of CWs stored at the last word location in the
'external eeprom and send it to the terminal program.
'then starting from the first byte, read every CWs(2 bytes each), one by one, and
'send it to the PC.
ReadEEprom:                      'first, read the value of total number of CW.
control = %01010111              'go to bank7.
addr    = %11111110              'to the last two bytes(the total number of commands).
GetTotalComd:
GoSub GetEEpromData              'get value of total number of commands. will be in
(B6,B7).
totalcmd = data                  'set total command to the value read.
SendTotalCmd:
GoSub SendPCData                 'send the total number of commands to pc
control = %01010000              'goto to bank0
addr = 0                         'start at 1st byte
ReadLoop:
IF totalcmd <= 0 Then Initialize'if there are no commands, start all over
```

```
GoSub GetEEpromData            'get the eeprom data. will be in(B6,B7)
GoSub SendPCData               'and send it to PC
IF addr = 0 Then RSwitchBank    'if 0-255-0 rolls over, time to switch bank
DoneRSB:
totalcmd = totalcmd - 1        'decrement totalcmd for each successful read
GoTo ReadLoop                  'read the next CW
RSwitchBank:                   'here, the banks are switched
control = control + 1          'just increment the last 3 bits
GoTo DoneRSB                   'go back

'-------------------------------------------------------------------------------------------------
'eeprom write program flow
'-------------------------------------------------------------------------------------------------

'Writing to external i2c EEPROM.
'first, get the value of total commands from the PC
'and write that value to the last word location in the eeprom.
'Then get each CW and write them, starting from the first byte in the eeprom.
WriteEEprom:
GetTotalCmd:
GoSub GetPCData                'get total commands from PC
totalcmd = data
control = %01010111            'go to bank7
addr    = %11111110            'to the last two bytes
WriteTotalCmd:
GoSub WriteEEpromData          'write the total command to eeprom last word location
control = %01010000            'switch back to bank0
addr    = 0                    'and to 1st byte location
WriteLoop:
IF totalcmd = 0 Then Initialize 'done writing. go back to main
GetData:
GoSub GetPCData
GoSub WriteEEpromData          'write the data to eeprom.
IF addr = 0 Then WSwitchBank    'when 0-255-0 rollovers, time to switch bank
DoneWSB:
totalcmd = totalcmd - 1        'decrement for each successful write
GoTo WriteLoop                 'loop for the next CW
WSwitchBank:                   '
control = control + 1          'increment the 3LSb for the next memory bank
GoTo DoneWSB                   'go back

'-------------------------------------------------------------------------------------------------
'Helper subroutines
'-------------------------------------------------------------------------------------------------

'receive the data from the PC serially.
```

```
'data will be in B6,B7
GetPCData:
SerIn DRX_J2TX,N9600,B6,B7
Return

SendPCData:
SerOut DTX_J2RX,N9600,(B6,B7)
Return

GetEEpromData:              'NOTE: data read is stored in (B6,B7).
I2CIN control,addr,B6        'first read the low byte
addr = addr + 1             'increment to the next byte
I2CIN control,addr,B7        'read the high byte
addr = addr + 1             'increment
Return                      'done reading so return

WriteEEpromData:             'NOTE: data in (B6,B7) are written.
I2COUT control,addr,(B6)      'write the low byte
Pause TWR                   'give eeprom 3ms to complete write(Twr)
addr = addr + 1             'increment to the next byte
I2COUT control,addr,(B7)      'write the high byte
Pause TWR                   'give eeprom 3ms to complete write(Twr)
addr = addr + 1             'increment to the next byte
Return                      'done writing so return
'-----------------------------------------------------------------------------------
```