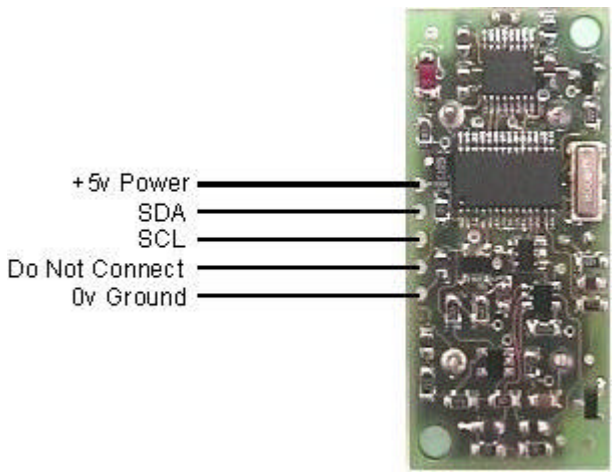
SRF08 Ultra sonic range finder

Technical Specification

Communication with the SRF08 ultrasonic rangefinder is via the I2C bus. This is available on popular controllers such as the OOPic and Stamp BS2p, as well as a wide variety of micro-controllers. To the programmer the SRF08 behaves in the same way as the ubiquitous 24xx series eeprom's, except that the I2C address is different. The default shipped address of the SRF08 is 0xE0. It can be changed by the user to any of 16 addresses E0, E2, E4, E6, E8, EA, EC, EE, F0, F2, F4, F6, F8, FA, FC or FE, therefore up to 16 sonar's can be used. In addition to the above addresses, all sonar's on the I2C bus will respond to address 0 - the General Broadcast address. This means that writing a ranging command to I2C address 0 (0x00) will start all sonar's ranging at the same time. This should be useful in ANN Mode (See below). The results must be read individually from each sonar's real address.

Connections

The "Do Not Connect" pin should be left unconnected. It is actually the CPU MCLR line and is used once only in our workshop to program the PIC16F872 on-board after assembly, and has an internal pull-up resistor. The SCL and SDA lines should each have a pull-up resistor to +5v somewhere on the I2C bus. You only need one pair of resistors, not a pair for every module. They are normally located with the bus master rather than the slaves. The SRF08 is always a slave - never a bus master. If you need them, I recommend 1.8k resistors. Some modules such as the OOPic already have pull-up resistors and you do not need to add any more.



Registers

The SRF08 appears as a set of 36 registers.

Location	Read	Write
0	Software Revision	Command Register
1	Light Sensor	Max Gain Register (default 31)
2	1st Echo High Byte	Range Register (default 255)
3	1st Echo Low Byte	N/A

~~	~~~~	~~~~
34	17th Echo High Byte	N/A
35	17th Echo Low Byte	N/A

Only locations 0, 1 and 2 can be written to. Location 0 is the command register and is used to start a ranging session. It cannot be read. Reading from location 0 returns the SRF08 software revision. By default, the ranging lasts for 65mS, but can be changed by writing to the range register at location 2. If you do so, then you will likely need to change the analogue gain by writing to location 1. See the **Changing Range** and **Analogue Gain** sections below.

Location 1 is the onboard light sensor. This data is updated every time a new ranging command has completed and can be read when range data is read. The next two locations, 2 and 3, are the 16bit unsigned result from the latest ranging - high byte first. The meaning of this value depends on the command used, and is either the range in inches, or the range in cm or the flight time in uS. A value of zero indicates that no objects were detected. There are up to a further 16 results indicating echo's from more distant objects.

Commands
The are three commands to initiate a ranging (80 to 82), to return the result in inches, centimeters or microseconds. There is also an ANN mode (Artificial Neural Network) mode which is described later and a set of commands to change the I2C address.

Command		Action
Decimal	Hex	
80	0x50	Ranging Mode - Result in inches
81	0x51	Ranging Mode - Result in centimeters
82	0x52	Ranging Mode - Result in micro-seconds
83	0x53	ANN Mode - Result in inches
84	0x54	ANN Mode - Result in centimeters
85	0x55	ANN Mode - Result in micro-seconds
160	0xA0	1st in sequence to change I2C address
165	0xA5	3rd in sequence to change I2C address
170	0xAA	2nd in sequence to change I2C address

Ranging Mode
To initiate a ranging, write one of the above commands to the command register and wait the required amount of time for completion and read as many results as you wish. The echo buffer is cleared at the start of each ranging. The first echo range is placed in locations 2,3. the second in 4,5, etc. If a location (high and low bytes) is 0, then there will be no further reading in the rest of the registers. The default and recommended time for completion of ranging is 65mS, however you can shorten this by writing to the range register before issuing a ranging command. Light sensor data at location 1 will also have been updated after a ranging command.

ANN Mode
ANN mode (Artificial Neural Network) is designed to provide the multi echo data in a way that is easier to input to a neural network, at least I hope it is - I've not actually done it yet. ANN mode provides a 32 byte buffer (locations 4 to 35 inclusive) where each byte represents the 65536uS maximum flight time divided into 32 chunks of 2048uS each -equivalentto about 352mm of range. If an echo is received within a bytes time slot then it will be set to no-zero, otherwise it will be zero. So if an echo is received from within the first 352mm, location 4 will be non-zero. If an object is detected 3m away the location 12 will be non-zero (3000/352 = 8) (8+4=12). Arranging the data like this should be better for a neural net than the other formats. The input to your network should be 0 if the byte is zero and 1 if its non-zero. I have a SOFM (Self Organizing Feature Map) in mind for the neural net, but will hopefully be useful for any type.

Location 4	Location 5	Location 6	Location 7	Locations 8 - 35
0 - 352mm	353 - 705mm	706 - 1057mm	1058 - 1410mm	and so on

Locations 2,3 contain the range of the nearest object converted to inches, cm or uS and is the same as for Ranging Mode.

Checking for Completion of Ranging
You do not have to use a timer on your own controller to wait for ranging to finish. You can take advantage of the fact that the SRF08 will not respond to any I2C activity whilst ranging. Therefore, if you try to read from the SRF08 (we use the software revision number a location 0) then you will get 255 (0xFF) whilst ranging. This is because the I2C data line (SDA) is pulled high if nothing is driving it. As soon as the ranging is complete the SRF08 will again respond to the I2C bus, so just keep reading the register until its not 255 (0xFF) anymore. You can then read the sonar data. Your controller can take advantage of this to perform other tasks while the SRF08 is ranging.

Changing the Range
The maximum range of the SRF08 is set by an internal timer. By default, this is 65mS or the equivalent of 11 metres of range. This is much further than the 6 metres the SRF08 is actually capable of. It is possible to reduce the time the SRF08 listens for an echo, and hence the range, by writing to the range register at location 2. The range can be set in steps of about 43mm (0.043m or 1.68 inches) up to 11 metres.
The range is ((Range Register x 43mm) + 43mm) so setting the Range Register to 0 (0x00) gives a maximum range of 43mm. Setting the Range Register to 1 (0x01) gives a maximum range of 86mm. More usefully, 24 (0x18) gives a range of 1 metre and 140 (0x8C) is 6 metres. Setting 255 (0xFF) gives the original 11 metres (255 x 43 + 43 is 11008mm). There are two reasons you may wish to reduce the range.
1. To get at the range information quicker
2. To be able to fire the SRF08 at a faster rate.
If you only wish to get at the range information a bit sooner and will continue to fire the SRF08 at 65ms of slower, then all will be well. However if you wish to fire the SRF08 at a faster rate than 65mS, you will definitely need to reduce the gain - see next section.
The range is set to maximum every time the SRF08 is powered-up. If you need a different range, change it once as part of your system initialization code.

Analogue Gain

The analogue gain register sets the *Maximum* gain of the analogue stages. To set the maximum gain, just write one of these values to the gain register at location 1. During a ranging, the analogue gain starts off at its minimum value of 94. This is increased at approx. 70uS intervals up to the maximum gain setting, set by register 1. Maximum possible gain is reached after about 390mm of range. The purpose of providing a limit to the maximum gain is to allow you to fire the sonar more rapidly than 65mS. Since the ranging can be very short, a new ranging can be initiated as soon as the previous range data has been read. A potential hazard with this is that the second ranging may pick up a distant echo returning from the previous "ping", give a false result of a close by object when there is none. To reduce this possibility, the maximum gain can be reduced to limit the modules sensitivity to the weaker distant echo, whilst still able to detect close by objects. The maximum gain setting is stored only in the CPU's RAM and is initialized to maximum on power-up, so if you only want do a ranging every 65mS, or longer, you can ignore the Range and Gain Registers.

Note - Effective in Ranging Mode only, in ANN mode, gain is controlled automatically.

Gain Register		Maximum Analogue Gain
Decimal	Hex	
0	0x00	Set Maximum Analogue Gain to 94
1	0x01	Set Maximum Analogue Gain to 97
2	0x02	Set Maximum Analogue Gain to 100
3	0x03	Set Maximum Analogue Gain to 103
4	0x04	Set Maximum Analogue Gain to 107
5	0x05	Set Maximum Analogue Gain to 110
6	0x06	Set Maximum Analogue Gain to 114
7	0x07	Set Maximum Analogue Gain to 118
8	0x08	Set Maximum Analogue Gain to 123
9	0x09	Set Maximum Analogue Gain to 128
10	0x0A	Set Maximum Analogue Gain to 133
11	0x0B	Set Maximum Analogue Gain to 139
12	0x0C	Set Maximum Analogue Gain to 145
13	0x0D	Set Maximum Analogue Gain to 152
14	0x0E	Set Maximum Analogue Gain to 159
15	0x0F	Set Maximum Analogue Gain to 168
16	0x10	Set Maximum Analogue Gain to 177
17	0x11	Set Maximum Analogue Gain to 187
18	0x12	Set Maximum Analogue Gain to 199
19	0x13	Set Maximum Analogue Gain to 212
20	0x14	Set Maximum Analogue Gain to 227
21	0x15	Set Maximum Analogue Gain to 245
22	0x16	Set Maximum Analogue Gain to 265
23	0x17	Set Maximum Analogue Gain to 288
24	0x18	Set Maximum Analogue Gain to 317
25	0x19	Set Maximum Analogue Gain to 352
26	0x1A	Set Maximum Analogue Gain to 395
27	0x1B	Set Maximum Analogue Gain to 450
28	0x1C	Set Maximum Analogue Gain to 524

29	0x1D	Set Maximum Analogue Gain to 626
30	0x1E	Set Maximum Analogue Gain to 777
31	0x1F	Set Maximum Analogue Gain to 1025

Note that the relationship between the Gain Register setting and the actual gain is not a linear one. Also there is no magic formula to say "use this gain setting with that range setting". It depends on the size, shape and material of the object and what else is around in the room. Try playing with different settings until you get the result you want. If you appear to get false readings, it may be echo's from previous "pings", try going back to firing the SRF08 every 65mS or longer (slower).
If you are in any doubt about the Range and Gain Registers, remember they are automatically set by the SRF08 to their default values when it is powered-up. You can ignore and forget about them and the SRF08 will work fine, detecting objects up to 6 metres away every 65mS or slower.

Light Sensor
The SRF08 has a light sensor on-board. A reading of the light intensity is made by the SRF08 each time a ranging takes place in either Ranging or ANN Modes (The A/D conversion is actually done just before the "ping" whilst the +/- 10v generator is stabilizing). The reading increases as the brightness increases, so you will get a maximum value in bright light and minimum value in darkness. It should get close to 2-3 in complete darkness and up to about 248 (0xF8) in bright light. The light intensity can be read from the Light Sensor Register at location 1 at the same time that you are reading the range data.

LED
The red LED is used to flash out a code for the I2C address on power-up (see below). It also gives a brief flash during the "ping" whilst ranging.

Changing the I2C Bus Address
To change the I2C address of the SRF08 you must have only one sonar on the bus. Write the 3 sequence commands in the correct order followed by the address. Example; to change the address of a sonar currently at 0xE0 (the default shipped address) to 0xF2, write the following to address 0xE0; (0xA0, 0xAA, 0xA5, 0xF2). These commands must be sent in the correct sequence to change the I2C address, additionally, No other command may be issued in the middle of the sequence. The sequence must be sent to the command register at location 0, which means 4 separate write transactions on the I2C bus. When done, you should label the sonar with its address, however if you do forget, just power it up without sending any commands. The SRF08 will flash its address out on the LED. One long flash followed by a number of shorter flashes indicating its address. The flashing is terminated immediately on sending a command the SRF08.

Address		Long Flash	Short flashes
Decimal	Hex		
224	E0	1	0
226	E2	1	1
228	E4	1	2
230	E6	1	3
232	E8	1	4
234	EA	1	5
236	EC	1	6

238	EE	1	7
240	F0	1	8
242	F2	1	9
244	F4	1	10
246	F6	1	11
248	F8	1	12
250	FA	1	13
252	FC	1	14
254	FE	1	15

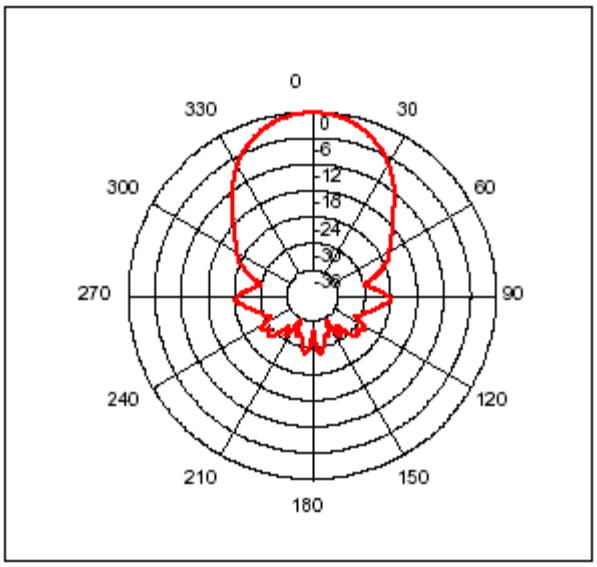
Take care not to set more than one sonar to the same address, there will be a bus collision and very unpredictable results.

Current Consumption
Average current consumption measured on our prototype is around 12mA during ranging, and 3mA standby. The module will automatically go to standby mode after a ranging, whilst waiting for a new command on the I2C bus. The actual measured current profile is as follows;

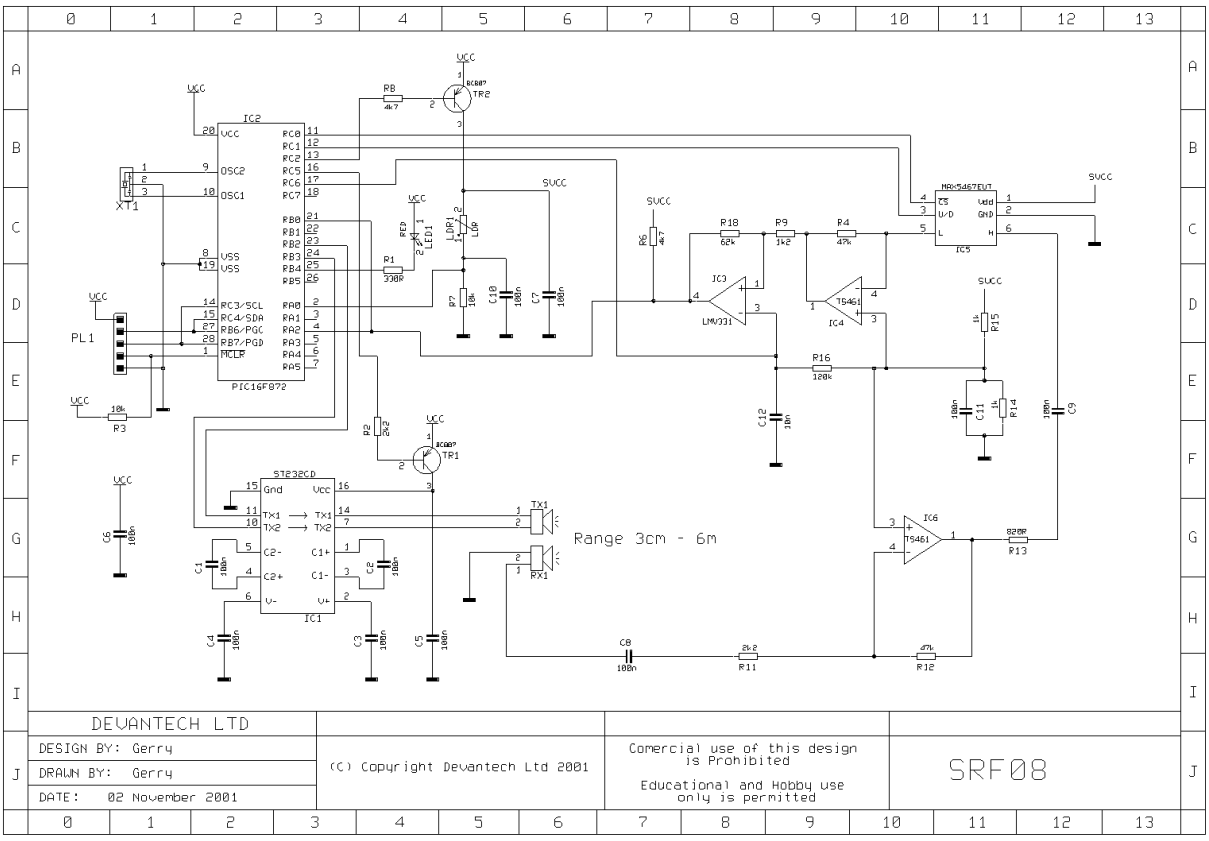
Operation	Current	Duration
Ranging command received - Power on	275mA	3uS
+/- 10v generator Stabilization	25mA	600uS
8 cycles of 40kHz "ping"	40mA	200uS
Ranging	11mA	65mS max
Standby	3mA	indefinite

The above values are for guidance only, they are not tested on production units.

Changing beam pattern and beam width
You can't! This is a question which crops up regularly, however there is no easy way to reduce or change the beam width that I'm aware of. The beam pattern of the SRF08 is conical with the width of the beam being a function of the surface area of the transducers and is fixed. The beam pattern of the transducers used on the SRF08, taken from the manufacturers data sheet, is shown below.



There is more information in the [sonar faq.]
Here you can have a look at the schematic



and here is the software

```
////////////////////////////////////////////////////////////////////////
//
//        SRF08 Ultrasonic rangefinder Software  - Preliminary
//
//        Written by Gerald Coe  - November 2001
//
// (C) Copyright Devantech Ltd 2001
//        Commercial use of this software is prohibited.
//        Private and Educational use only is permitted
//
////////////////////////////////////////////////////////////////////////
//
// Sonar uses one of 16 addresses -> 0xe0 - 0xfe
// Bit 0 is always zero - its the i2c rd/wr bit
//
////////////////////////////////////////////////////////////////////////
//
// This software is written for the HITECH PICC C compiler
//
////////////////////////////////////////////////////////////////////////


#include"pic.h"

#define version        1                       // software version
#define echo           RA2             // 1st stage echo line
#define led                    RB4             // low to light led
#define pot_ud                 RC0             // Pot up/dw control
#define pot_cs         RC1             // Pot chip select control
#define anpower        RC2             // analog power - low on
#define txpower        RC5             // Tx power - low on
#define clamp          RC6             // comparator clamp
#define clamp_en       TRISC6          // comparator clamp enable
#define detect         RC7


// initialise the eeprom with 0xea i2c address
// the default shipping address is 0xe0, our test jig
// will change the address to 0xe0
__EEPROM_DATA (0xff, 0xff, 0xff, 0xff, 0xff, 0xea, 0xff, 0xff);


//prototypes
voidsetup(void );
void burst(void);
void multi_range(void);
void ann_range(void);
void set_bit(unsigned char idx);
```

```
void flash_addr(void);
void convert(unsigned char cmd, unsigned char idx);


// global variables
char buffer[36];
char loop, dlyctr;
bit timeout;
unsigned char command, index;
unsigned char gain, gaincnt;


// the interrupt
void interrupt the_only_one(void)
{
static char idx=0, wr_addr=0;
char i2c_data;

        if(SSPIF) {                                             // I2C interrupt
                SSPIF = 0;

                if(!STAT_DA) {                                  // low = address
                        wr_addr=0;
                }
                if(STAT_RW) {                                   // high = read from this
program
                        SSPBUF = buffer[idx];           // send data
                        if(idx<36) ++idx;                       // limit index to 32 bytes
                        CKP = 1;                                                // release I2C clock
line
                }
                else {
                        i2c_data = SSPBUF;              // read incoming data
                        wr_addr++;
                        if(wr_addr==2) {                        // 1st byte written is internal
location
                                idx = i2c_data;                 // lower 4 bits only (0-35 index)
                                if(idx>35) idx=35;      // limit index
                        }
                        else {
                                if(idx==0 && wr_addr==3) { // register 0 is start ping command
                                        command=i2c_data;
                                }
                        }
                }
                SSPOV = 0;
        }

        if(TMR1IF==1) {                         // timer1 is the echo timer
                timeout = 1;                    // end of echo timing when it rolls over
                TMR1ON = 0;
```

```
                TMR1IF = 0;
        }
}



void main(void)
{
static unsigned char seq=0;

        setup();                                        // initialise the peripherals
        flash_addr();                   // flash the I2C address on LED

        while(1) {
                while(!command);        // wait for start command

                timeout = 1;            // end of echo timing when new command arrives
                TMR1ON = 0;
                TMR1IF = 0;

                switch(command) {
                        case 0x00:              // Gain commands to limit max. gain in
                        case 0x01:              // Range Mode
                        case 0x02:
                        case 0x03:
                        case 0x04:
                        case 0x05:
                        case 0x06:
                        case 0x07:
                        case 0x08:
                        case 0x09:
                        case 0x0A:
                        case 0x0B:
                        case 0x0C:
                        case 0x0D:
                        case 0x0E:
                        case 0x0F:
                        case 0x10:
                        case 0x11:
                        case 0x12:
                        case 0x13:
                        case 0x14:
                        case 0x15:
                        case 0x16:
                        case 0x17:
                        case 0x18:
                        case 0x19:
                        case 0x1A:
                        case 0x1B:
                        case 0x1C:
                        case 0x1D:
```

```
                case 0x1E:
                case 0x1F:      gain = command;
                                break;

                case 0x80:                                              //
inches, centimetres or uS
                case 0x81:
                case 0x82:      multi_range();          // 2byte multi-ping data
                                seq = 0;
        // reset address change sequence
                                break;
                case 0x83:
                case 0x84:
                case 0x85:      ann_range();            // 2byte 1st, 1byte multi-
pings
                                seq = 0;
        // reset address change sequence
                                break;
                case 0xa0:      seq = 1;                                // start of
sequence to change address
                                break;
                case 0xaa:      if(seq==1) ++seq;       // 2nd of sequence to
change address
                                else seq = 0;
                                break;
                case 0xa5:      if(seq==2) ++seq;       // 3rd of sequence to change
address
                                else seq = 0;
                                break;
                case 0xe0:                      // if seq=3 user is changing sonar I2C
address
                case 0xe2:
                case 0xe4:
                case 0xe6:
                case 0xe8:
                case 0xea:
                case 0xec:
                case 0xee:
                case 0xf0:
                case 0xf2:
                case 0xf4:
                case 0xf6:
                case 0xf8:
                case 0xfa:
                case 0xfc:
                case 0xfe:      if(seq==3) {
                                        EEPROM_WRITE(5, command);
                                        SSPADD = command;
                                        led = 0;
                                }
                                seq = 0;
```

```
                                                        break;
                }

                command = 0;
                anpower = 1;                                                    // analog
power off
        }
}



////////////////////////////////////////////////////////////////////////

// The burst routine generates an acurately timed 40khz burst of 8 cycles.
// Timing assumes an 8Mhz PIC (500nS instruction rate)
// I drop down to assembler here because I don't trust the compiler to
// always generate accurately timed code with different versions or
// optimisation settings
//
void burst(void) {

char x;
        clamp = 0;
        clamp_en = 0;                   // force low on clamp line

        pot_cs = 1;                             // deselect pot
        led = 0;                                        // on
        GIE = 0;                                        // disable interrupts for timing
accuracy
        txpower = 0;                    // turn st232 on
        anpower = 0;                    // turn analog power on
        loop = 8;                               // number of cycles in burst
        pot_ud = 1;                             // select pot inc mode
        x = 0;
        while(--x);                             // wait for +/- 10v to charge up.
        pot_cs = 0;                             // enable pot
        for(x=2; x<36; x++) {           // and take opportunity to clear echo buffer
                pot_ud = 0;                     // and reset pot wiper
                buffer[x] = 0;
                pot_ud = 1;
        }
        clamp_en = 1;                   // release clamp line

        ADGO = 1;                               // convert light sensor
        pot_cs = 1;                             // deselect pot
        while(ADGO);
        pot_ud = 0;                             // select pot dec mode
        buffer[1] = ADRESH;             // store light sensor reading

#asm
burst1: movlw           0x14                    ; 1st half cycle
```

```
                        movwf       _PORTB
                        nop

                        movlw       7                       ; (7 * 3inst * 500nS) -
500nS = 10uS
                        movwf       _dlyctr         ; 10uS + (5*500nS) = 12.5uS
burst2: decfsz  _dlyctr,f
                        goto        burst2

                        movlw       0x18                ; 2nd half cycle
                        movwf       _PORTB

                        movlw       6                       ; (6 * 3inst * 500nS) -
500nS = 8.5uS
                        movwf       _dlyctr         ; 8.5uS + (8*500nS) = 12.5uS
burst3: decfsz  _dlyctr,f
                        goto        burst3
                        nop
                        decfsz _loop,f
                        goto        burst1

                        movlw       0x10                ; set both drives low
                        movwf       _PORTB
#endasm
        GIE = 1;
        txpower = 1;                    // turn st232 off
                                                // Led off
        led = 1;
        pot_cs = 0;                             // enable pot
}


//////////////////////////////////////////////////////////////////////////

void multi_range(void) {

unsigned char tone_cnt, period, cmd;

        burst();                        // send 40khz burst, reset pot wiper and clear buffer

        cmd = command;          // save cmd so we know how to convert result
        TMR0 = 0;
        TMR1H = 0;
        TMR1L = 0;
        timeout = 0;
        tone_cnt = 3;
        index = 2;
        TMR1ON = 1;
        TMR2 = 0;
        TMR2IF = 0;
        gaincnt = gain;
```

```
        while(timeout==0) {                                     // while still timing
stage3
                while(timeout==0 && echo==0) {          // wait for high
                        if(TMR2IF && gaincnt) {
                                pot_ud = 1;
                                --gaincnt;
                                TMR2IF = 0;
                                pot_ud = 0;
                        }
                }
                while(timeout==0 && echo==1) {          // wait for low
                        if(TMR2IF && gaincnt) {
                                pot_ud = 1;
                                --gaincnt;
                                TMR2IF = 0;
                                pot_ud = 0;
                        }
                }

                if(timeout==0) {
                        period = TMR0;
                        TMR0 = 0;
                        if(period>40 && period<60) {
                                if(!(--tone_cnt)) {
                                        do {
                                                buffer[index] = TMR1H;
                                                buffer[index+1] = TMR1L;
                                        }while(buffer[index] != TMR1H);

                                        convert(cmd, index);            // convert to in, cm or uS
                                        if(index == 36) return;
                                        index += 2;
                                        tone_cnt = 3;
                                        period = 0;
                                        while(--period){                                //delay
about 5 inches of range
                                                if(TMR2IF && gaincnt) {
                                                        pot_ud = 1;
                                                        --gaincnt;
                                                        TMR2IF = 0;
                                                        pot_ud = 0;
                                                }
                                        }
                                        while(--period){
                                                if(TMR2IF && gaincnt) {
                                                        pot_ud = 1;
                                                        --gaincnt;
                                                        TMR2IF = 0;
                                                        pot_ud = 0;
                                                }
                                        }
```

```
                                        while(--period){
                                                if(TMR2IF && gaincnt) {
                                                        pot_ud = 1;
                                                        --gaincnt;
                                                        TMR2IF = 0;
                                                        pot_ud = 0;
                                                }
                                        }
                                }
                        }
                        else tone_cnt=3;
                }
        }
}

//////////////////////////////////////////////////////////////////


void ann_range(void) {

unsigned char tone_cnt, period, index, cmd;

        burst();                        // send 40khz burst and clear buffer

        cmd = command;                  // save cmd so we know how to convert result
        TMR0 = 0;
        TMR1H = 0;
        TMR1L = 0;
        timeout = 0;
        tone_cnt = 3;
        index = 2;
        TMR1ON = 1;

        while(timeout==0) {                                             // while still timing
stage3
                while(timeout==0 && echo==0) {          // wait for high
                        if(TMR2IF) {
                                pot_ud = 1;
                                TMR2IF = 0;
                                pot_ud = 0;
                        }
                }
                while(timeout==0 && echo==1) {          // wait for low
                        if(TMR2IF) {
                                pot_ud = 1;
                                TMR2IF = 0;
                                pot_ud = 0;
                        }
                }
```

```
                if(timeout==0) {
                        period = TMR0;
                        TMR0 = 0;
                        if(period>40 && period<60) {
                                if(!(--tone_cnt)) {
                                        set_bit(TMR1H);
                                        if(index==2) {                  // only 1st echo in
ann mode
                                                do {
                                                        buffer[index] = TMR1H;
                                                        buffer[index+1] = TMR1L;
                                                }while(buffer[index] != TMR1H);
                                                convert(cmd, index);    // convert to in, cm
or uS
                                                index += 2;
                                        }
                                        tone_cnt = 1;                   // to detect
continuing echo
                                }
                        }
                        else tone_cnt=3;
                }
        }
}



void set_bit(unsigned char idx)
{
char pos;

        pos = idx&7;                    // lower 3 bits indicate bit position
        idx = (idx>>3)+4;               // index into buffer
        switch(pos) {
                case 0: buffer[idx] |= 0x01;
                                        break;
                case 1: buffer[idx] |= 0x02;
                                        break;
                case 2: buffer[idx] |= 0x04;
                                        break;
                case 3: buffer[idx] |= 0x08;
                                        break;
                case 4: buffer[idx] |= 0x10;
                                        break;
                case 5: buffer[idx] |= 0x20;
                                        break;
                case 6: buffer[idx] |= 0x40;
                                        break;
                case 7: buffer[idx] |= 0x80;
                                        break;
        }
```

```
}

////////////////////////////////////////////////////////////////

void convert(unsigned char cmd, unsigned char idx)
{
unsigned int x;

        x = (buffer[idx]<<8) + buffer[idx+1];
        switch(cmd) {
                case 0x80:
                case 0x83:      x /= 148;       // convert to inches
                                                break;
                case 0x81:
                case 0x84:      x /= 58;                // convert to cm
        }
        buffer[idx] = x>>8;
        buffer[idx+1] = x&0xff;                 // replace uS with inches, cm or uS
}


////////////////////////////////////////////////////////////////

void setup(void)
{
//      _CONFIG(0x0d42);                // code protected, hs osc
        __CONFIG(0x3d72);               // code not protected, hs osc

        ADCON1 = 0x0e;                          // PortA 0 is analog, rest are digital

        ADCON0 = 0x41;                          // convert ch0
        PORTC = 0xff;                           // nothing powered at start
        TRISA = 0xff;                   // All inputs
        TRISB = 0xc3;                           // 11000011 PB7,6,1,0 are inputs, rest are
outputs
        TRISC = 0x18;                           // 00011000 RC3,4 are inputs
        OPTION = 0x08;                          // portb pullups on, prescaler to wdt
        T1CON = 0x10;                           // timer1 prescale 1:2, but not started yet
        T2CON = 0x04;                           // 1:4 prescale and running
//      T2CON = 0x06;                           // 1:16 prescale and running
        PR2 = 140;                              // set TMR2IF every 280uS at 8MHz
        SSPSTAT = 0x80;                 // slew rate disabled
        SSPCON  = 0x36;                 // enable port in 7 bit slave mode
        SSPCON2 = 0x80;                 // enable general call (address 0)
        SSPADD  = EEPROM_READ(5);       // address 0xE0 - 0xFE
        if(SSPADD<0xE0)
                SSPADD=0xE0;                    // protection against corrupted eeprom
```

```
        else SSPADD &= 0xfe;
        buffer[0] = version;            // software revision
        SSPIE = 1;                              // enable I2C interrupts
        TMR1IE = 1;                             // enable timer1 interrupts
        TMR1IF = 0;
        SSPIF = 0;
        PEIE = 1;                               // enable peripheal interrupts
        GIE = 1;                                        // enable global interrupts
        gain = 32;                              // ma ximum gain at power-up
}

//////////////////////////////////////////////////////////////////

void flash_addr(void)
{
unsigned char count;
long delay, on, off;

        on = off = 30000;

        count = ((SSPADD>>1)&0x0f)+1;
        do {
                delay = on;
                on = 10000;
                led = 0;                                        // led on
                while(--delay) if(command) return;
                delay = off;
                off = 20000;
                led = 1;                                        // led off
                while(--delay) if(command) return;
        }
        while(--count);
}

//////////////////////////////////////////////////////////////////
```