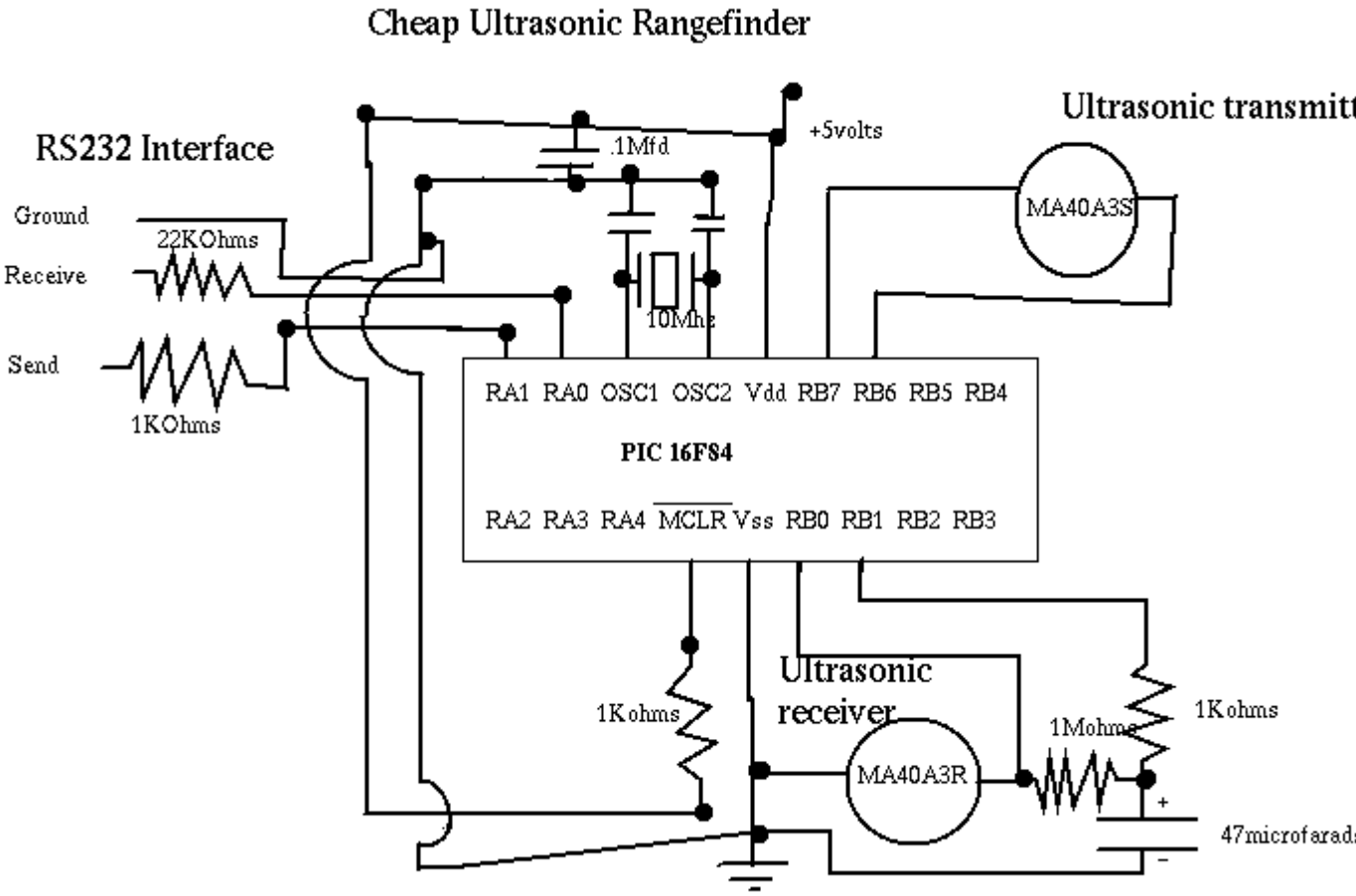
Cheap Sonar Range Finder Design

Description

Everything is controlled by a PIC16F84. The circuit has very few parts, and the parts are cheap.

The quit description. Send out 16 pulses at 40Khz through one ultrasonic transducer. Time the echos as they come back. Send the time in ASCII out the serial port at 9600 baud. Repeat.



Parts list

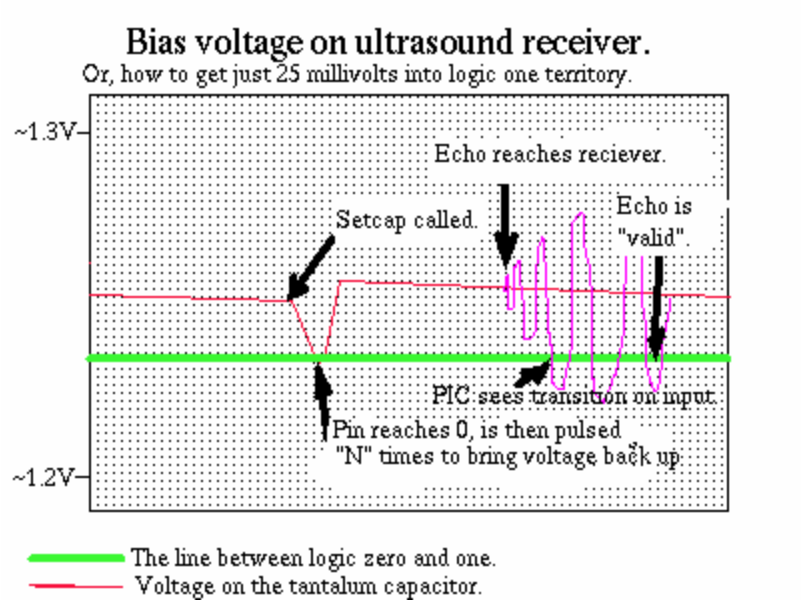
1. 1 **Microchip PIC 16F84**: an 18 pin cpu with 68 bytes of RAM and 1024 words of on-chip EEPROM program memory.
2. 1 **muRata MA40A3R ultrasonic receiver** The reciever and transmitter were $2.50 a pair from a surplus catalog. I don't know any of the specs on these, I just guessed. I originally used Mouser part number 251-1603 Ultrasonic Transducers,

but these are $6.20 each. And I managed to fry one in a overly powerfull circuit. *Oops*

3. 1 **muRata MA40A3S ultrasonic transmitter**.
4. 1 **1 KOhm resistor** Limits current on RB1 charging the capacitor. This resistor can be anything greater than 250 Ohms.
5. 1 **1 KOhm resistor** Limits current on RS232 Send. This protects the PIC if the RS232 cable is wired up wrong. It's not needed for folks who always get their cables right the first time.
6. 1 **1 KOhm resistor** Pullup resistor for MCLR, Master Clear, what other chips call "Reset". You can reset the PIC by shorting this pin to ground. The resistor limits the current when the pin is shorted to ground. This resistor is optional. If MCLR is tied directly to **Vdd**, then you have use the power switch to reset the circuit.
7. 1 **22 KOhms resistor** Limits current from the remote RS232 line to something that the PIC input pin can safely handle. Any value from 10 KOhms to 1MOhm will probably work. Really high values make the circuit susceptible to noise.
8. 1 **1 MOhm resistor** This biases the ultrasonic receiver pin to the same voltage as the tantalum capacitor, without putting much load on the receiver. Any value from 50KOhms on up will probably work.
9. 1 **.1 Microfarad capacitor** This should be connected as closely as possible to the Vdd and Vss pins of the PIC. Which is to say, the circuit on my breadboard had a lot of noise when the capacitor was near the power supply and away from the PIC.
10. 1 **47 Microfarad tantalum capacitor**, a 1 Microfarad disk capacitor was too small, or had too much leakage. This size capacitor worked. Really large values will slow down the circuit.
11. 1 **10Mhz crystal** see the PIC databook for crystal selection rules (in other words, I've forgotten what type of crystal it is.)
12. 2 **small capacitors** these capacitors are the right size to let the PIC drive the 10Mhz crystal. I figured out the right size long ago. I have a bunch of them with matching crystals. I can't read the little tiny writing on the capacitors to see the value. *Oops.*

Theory

A PIC input pin has a really high impedance. It switches at something around 1.2 volts. So, a PIC input pin is all you need to sense a signal of a few dozen millivolts, provided you can bias the signal to near the switch point.



First, pretend I did a good job drawing a sinewave. Also realize that the picture is not to scale.

The thick green line is at the transition between zero and one. That transition happens to be at 1.23volts in my picture, but it slowly changes. I doesn't change much in 3 milliseconds. It does vary part to part, and with temperature and probably many other factors. The circuit self calibrates before each reading. That voltage can drift without affecting operation. Likewise, the resistors and capacitors in the circuit can drift. The circuit doesn't need precision components, or manual adjustment.

Ok, the "N" in N pulses is adjustable. But that is in software and can be changed over the serial port.

Transmitter

The ultrasonic transmitter is connected directly to pins RB7 and RB6 of the PIC. The pins are left as inputs except when actually transmitting. To transmit, the pins are set to outputs and driven at 40Khz 180 degrees out of phase, ie. RB7 is a logic one whenever RB6 is a logic zero and vice versa.

A 40Khz squarewave has 80,000 transitions per second. This is one transition every 12.5 microseconds. With a 10 Mhz crystal, the PIC needs to make one transition every 31.2 cycles. Reality gets in the way. The code actually does one transition every 31 cycles, Making the output frequency be 40.3Khz.

I arbitrarily chose 16 cycles. This works, so I haven't experimented with other values.

Here is the code for the output routine "`osc16`". Copies of TRISB and the output bits of PORTB are kept in file registers `shadowtb` and `shadowb` respectively. The loop counter is

kept in the file register `ocnt`. The subroutine `del26` takes exactly 26 cycles, including the call and return statement.

Instructions that change the program counter take 2 cycles, the rest take 1 cycle. So a `nop` instruction takes 1 cycle, jumping to the next instruction, `goto $+1`, takes 2 cycles.

```
UTX1    equ     7
UTX2    equ     6

osc16
        movlw   16   ; 16 cycles at 40Khz
        movwf   ocnt
        movlw   1
        bsf     shadowb,UTX1    ;
        bcf     shadowb,UTX2    ; This pin is the negation of UTX1
        bcf     shadowtb,UTX1
        bcf     shadowtb,UTX2
        movf    shadowtb,w
        tris    PORTB
oloop                           ;0
        movf    shadowb,w       ;1
        movwf   PORTB           ;2
        call    del26           ;28
        movf    shadowb,w       ;29
        xorlw   (1<<UTX1)|(1<<UTX2) ;30
        goto    $+1             ;31
        movwf   PORTB           ;33   Should be 31 cycles between
writes to PORTB
        call    del26           ;59
        decfsz  ocnt,F          ;60
        goto    oloop           ;61/62
        bsf     shadowtb,UTX1
        bsf     shadowtb,UTX2
        movf    shadowtb,w
        tris    PORTB
        return
```

Receiver

In the real circuit, the transducers are on foot long cables. The transducer naturally filters out everything that isn't close to 40Khz. The cables are not so selective. The cables act like antennas. Rather than upgrade to fancier cables, the receive routine uses a crude digital filter.

The input routine wants to see an alternating like pattern 10101 when it samples at 80Khz. That scheme will fail if the samples are close to the zero crossing of the input wave. The input routine actually samples at 160Khz, keeping 2 independent counters, `hits1` and `hits2`. If either counter sees 4 transitions at 80Khz, the routine returns success. I check to see if a counter has reached 4 by checking bit number 2 of the counter.

The loop is 31 cycles long, 12.4 microseconds, one half of a 40.3Khz wave. The receiver routine counts each time through the loop. After 255 passes through the loop, it gives up.

The speed of sound is about 1 foot per millisecond. This system times the round trip of the sound pulse. If there is an object 1 foot away, the sound will take a millisecond to get there, and a millisecond to come back, for a total of 2 milliseconds. That is a count of about 160. So the maximum range of this software is only 18 inches. The drive circuit is wimpy enough, that 18 inches is also about the range of the hardware.

```
echotime ; Time until we see an echo.
        clrf    ecnt
        clrf    hitsa
        clrf    hitsb
          ; This loop is exactly 31 cycles long, which is half of a
40Khz cycle
echoa                               ; 0
        btfsc   flags,LASTA         ; 1
        goto    lasta1              ; 2/3

lasta0                                      ;2
        nop      ; equalize path length  ;3
        btfss   PORTB,URX               ;4
        goto    samea                   ;5/6
        bsf     flags,LASTA             ;6
        goto    diffa                   ;8
lasta1                                      ;3
        btfsc   PORTB,URX               ;4
        goto    samea                   ;5/6
        bcf     flags,LASTA             ;6
        goto    diffa                   ;8
samea                                       ;6
        clrf    hitsa                   ;7
        call    del4                    ;11
        goto    echob                   ;13
diffa                                       ;8
        incf    hitsa,f                 ;9
        btfsc   hitsa,2                 ;10   Did we see 2 whole cycles
(4 half cycles)
        goto    echofound               ;11/12
        goto    $+1                     ;13
echob                                       ;13
        goto    $+1                     ;15
        btfsc   flags,LASTB             ;16
        goto    lastb1                  ;17/18
lastb0                                      ;17
        nop      ; equalize path length ;18
        btfss   PORTB,URX               ;19
        goto    sameb                   ;20/21
        bsf     flags,LASTB             ;21
        goto    diffb                   ;23
lastb1                                      ;18
        btfsc   PORTB,URX               ;19
        goto    sameb                   ;20/21
        bcf     flags,LASTB             ;21
        goto    diffb                   ;23
sameb                                       ;21
        clrf    hitsb                   ;22
        call    del4                    ;26
```

```
        goto    next                    ;28
diffb                                   ;23
        incf    hitsb,f                 ;24
        btfsc   hitsb,2                 ;25    Did we see 2 whole
cycles (4 half cycles)
        goto    echofound               ;26/27
        goto    $+1                     ;28

next                                    ;28
        incfsz  ecnt,F                  ;29
        goto    echoa                   ;30/31
        movlw   255
        return

echofound;
        movf    ecnt,w
        return
```

The capacitor bias code

Here is the software for biasing the capacitor. I still need to write some notes on it. The entry point is `setcap`, near the bottom. Note that `pulseconstant` is set by a command over the serial port. I hope to automate away that kluge.

```
pulsecap  ;             make the cap pin an output for 3 cycles.
        bcf     shadowtb,CAP
        movf    shadowtb,w
        tris    PORTB
        bsf     shadowtb,CAP
        movf    shadowtb,w
        tris    PORTB
        return
longpulsecap  ;         make the cap pin an output for 29 cycles.
        bcf     shadowtb,CAP
        movf    shadowtb,w
        tris    PORTB
        call    del26
        bsf     shadowtb,CAP
        movf    shadowtb,w
        tris    PORTB
        return

capto0
        btfss   PORTB,CAP  ; loop until the pin is at 0
        return
        bcf     shadowb,CAP
        movf    shadowb,w
        movwf   PORTB
        call    pulsecap
        goto    capto0
capto1
        btfsc   PORTB,CAP  ; loop until the pin is at 1 on input
        return
        bsf     shadowb,CAP   ; Make cap pin a 1 when output
        movf    shadowb,w
        movwf   PORTB
```

```
        call    pulsecap
        goto    capto1

setcap  ;      Charge up the cap until it is right at the transition
from 0 to 1
        call    capto1
        call    capto0
        call    capto1
        call    capto0
        bsf     shadowb,CAP   ; Make cap pin a 1 when output
        movf    shadowb,w
        movwf   PORTB
        movf   pulseconstant,w
        movwf   ncnt
setcapl
        call    longpulsecap
        decfsz  ncnt,f
        goto    setcapl
        return
```

What next?

The next thing to do is to add 3 more pairs of ultrasonic transducers. The other transmitters can share RB7, so it takes 5 pins for 4 transmitters. Likewise, the capacitor can be shared among the receivers, so they only need 5 pins for 4 receivers. The serial port takes 2 pins, so there is still one pin left free.

Right now, the system checks for a start bit in the delay routines (`del26` in the above code snippits). The system should also be check during the `echotime` routine.

My hardware is pretty icky right now. The receiver sees a weak echo through the mechanical transducer mounting. This is the real limit on range. If I get better mechanical mounting for the transducers, I can improve the range dramatically.

I could get much longer range with a more powerfull driver circuit. I wanted to prove that I didn't need all that stuff. Now that I've proved it, I can add some oomph.

I'm open to other suggestions.