This is a tutorial about controling the HC-SR04 Ultrasonic Sensor . See http://jaktek.com/?page_id=87 for information about this sensor. Read the datasheet.

The HC-SR04 has 4 pins clearly marked, vcc,trig,echo,gnd. The operation is: send a short, but long enough 10+us, high pulse on the trigger pin. Wait for a short indeterminate length of time for the echo line to go high.

Time the length of time the line stays high.

Don't send trigger pluses too often. From the data sheet “the recommend cycle period should be no less than 50ms”

I wanted to control this sensor with the Atmega328P, because I had one on the Arduino UNO board. I had done a little work earlier with the UNO and Adafruit Motorshield. So I wanted to use resourses not needed by the shield. I chose to use timer1, the 16 bit timer, The int0 pin aka PORTD2, or pin2 on the UNO.

The HC-SR04 only needs a trigger and an echo line, actually over at JAKTEK they show how only 1 line is needed.

I wanted to write in AVR assembler so the only tools I used were the Atmel Studio 6, and Avrdude which comes with WinAVR, both free, Also needed a terminal emulator to run in my PC so I could see some output. I used PuTTY because I have used it before and still had it installed. You need to know the com port, and the settings, here I use 9600,8,1. The binary to ASCII conversion comes from (C)2002 by http://www.avr-asm-tutorial.net , though I made some minor changes. Changing code is a way to learn. This ASCII conversion is from a previous exercise. As is the send routine.

Analyzing the problem:

Need to send a 10+ us pulse to the trigger. Need to wait for the echo line to go high and time how long until it drops. The send the result to my PC. In addition, don't want to wait forever. My solution depends on a timer and interrupts. You should read the Avrfreaks timer tutorial [TUT] [C] Newbie's Guide to AVR Timers and the interrupt tutorial [TUT] Newbie's Guide to AVR Interrupts .

So I thought in terms of states :

zero state: ready to send the trigger.

One: Set the trigger high, waiting at least 10us before setting it back low. Go to 2

Two: Done with trigger, wait for echo pin to go high. Go to 3

Three: Echo pin high, time how long it stays high. Go to 4

Four: Echo pin goes low. Go to 7

Five: Echo pin doesn't go low in time Go to 0

Six: Error state, go to 0

Seven: Got a time, convert and send to PC. Go to 0

A note here about time: thinking about secs, ms,us makes my head spin. So with the help of Wikipedia and the OpenOffice spreadsheet. I converted all times to seconds.

Seconds	1
millie ms	0.001
micro us	0.000001
nano ns	0.000000001
10us	0.00001
16Mhz period	0.0000000625 (1/16000000)
10us in clks	160 which is equal to 0x00A0

since I want to run the timer to TOV, I will start the CLK/1 with 65535-160 = 0xFF5F, but actually I want a little head room, so I used 0xFF24. The point here is that the 16 bit timer will work just fine. By presetting the timer near the top, it will count to the top and overflow. The TOV interrupt will be the signal to drop the trigger.

Next I estimate how long a time span needed for the echo. Over at JAKTEK they do a lot of analyzing, so go take a look http://jaktek.com/?page_id=87 ..

Speed of sound at sea level, from Wikipedia "In dry air at 20 °C (68 °F), the speed of sound is 343.2 metres per second ". But the speed of sound varies by altitude and temperature. According to http://www.engineeringtoolbox.com it is about 335.5 where I am (1250 m). So I decided to not bother converting from the clocks.

The important issue is will the timer reach TOV while waiting for the echo. Since the datasheet for the HC-SR04 only claims 500cm, or a 10 m round trip, lets see how many clocks cycles are needed. 343M/sec => .1/343 = .00291545 sec/m *10 .0291545, divide by time of 16Mhz clock = 466472 cycles, not a fit, but the clk/8 = 58309, so we can use clk/8 and if the timer hits TOV we are out of range for the sensor.

I chose the INT0 external interrupt because the one interrupt can be triggered by both a falling and a rising edge.

Thus the resourses needed are:

a timer, 16-bit is a good fit, but can be done with an 8 bit.
INT0 or INT1
PC5 for the trigger

Make sure you have the correct interrupt jump table. This is for the Atmega328P.   Refer to the datasheet for the device you will use.

```
.org 0x0000    ;Places the following code from address 0x0000

                jmp RESET ; Reset Handler
                jmp EXT_INT0 ; IRQ0 Handler
                jmp EXT_INT1 ; IRQ1 Handler
                jmp PCINT0L ; PCINT0 Handler
                jmp PCINT1L ; PCINT1 Handler
                jmp PCINT2L ; PCINT2 Handler
                jmp WDT ; Watchdog Timer Handler
                jmp TIM2_COMPA ; Timer2 Compare A Handler
                jmp TIM2_COMPB ; Timer2 Compare B Handler
                jmp TIM2_OVF ; Timer2 Overflow Handler
                jmp TIM1_CAPT ; Timer1 Capture Handler
                jmp TIM1_COMPA ; Timer1 Compare A Handler
                jmp TIM1_COMPB ; Timer1 Compare B Handler
                jmp TIM1_OVF ; Timer1 Overflow Handler
                jmp TIM0_COMPA ; Timer0 Compare A Handler
                jmp TIM0_COMPB ; Timer0 Compare B Handler
                jmp TIM0_OVF ; Timer0 Overflow Handler
                jmp SPI_STC ; SPI Transfer Complete Handler
                jmp USART_RXC ; USART, RX Complete Handler
                jmp USART_UDRE ; USART, UDR Empty Handler
                jmp USART_TXC ; USART, TX Complete Handler
                jmp ADCR ; ADC Conversion Complete Handler
                jmp EE_RDY ; EEPROM Ready Handler
                jmp ANA_COMP ; Analog Comparator Handler
                jmp TWI ; 2-wire Serial Interface Handler
                jmp SPM_RDY ; Store Program Memory Ready Handler
;
RESET: ldi r16, high(RAMEND)            ; Main program start
                out SPH,r16             ; Set Stack Pointer to top of RAM
                ldi r16, low(RAMEND)
                out SPL,r16
                cli                     ; disable interupts

;       Setup the 16-bit timer to time the trigger pulse:

                                        ; set up timer1

                clr r16                 ;
                sts TCCR1A, r16         ;  normal mode timer
                sts TCCR1C, r16         ;  just set zero, using normal mode
                ldi r16, 0b00000001     ;
                sts TIMSK1, R16         ; overflow inturrupt enable

;       Next INT0

                                        ; int0 interupt PD2
                cbi DDRD, 2             ; set for input
                ldi r16, 0b00000001
                sts EICRA, r16          ; any change on int0 to trigger interrupt
                out EIMSK, r16          ; enble interrupt on int0
                sbi DDRC, 5             ; output Portc5, Arduino pin Analog in 5, trigger the HC-SR04
                sei                     ;enable interrupts
```

```
;        Use GPIOR1 for state variable.


                clr r16
                sts GPIOR1, r16          ;   state variable, to state 0

;        Make sure timer is off

                ldi r16, 0b00000000   ;
                sts TCCR1B, r16          ; turn off timer


;        Start the timer in  subroutine setst.

        loop:
                lds r16, GPIOR1           ; get state
                cpi r16, 0
                brne loop8
                rcall setst              ; start the trigger and move to state 1, note we don't block the main loop
                                         ;just start the timer,
                rjmp loop
        loop8:
                clr r16
                sts GPIOR1, r16          ; for test just go back to state 0
                rcall delay_05
                rjmp loop



        setst:
                cli                      ;turn off interupts when reading or writing the two byte registers
                clr r27                  ; test value, so the led stays on long enough to see  will be 0xFF
                clr r26                  ;test value, will be 0x24 with a 16Mhz clock
                sts TCNT1H,r27            ; high byte  Order write (and reading) to 16 bit registers must be done in
                sts TCNT1L,r26           ; proper order, see datasheet
                sei                      ;turn interupts back on
                ldi r16, 0b00000001
                sts TCCR1B, r16          ; start the timer with clk 1, timer will run to overflow, so pulse > 10us.
                sbi PORTC,5              ;, start the trigger
                sts GPIOR1, R16           ; set state 1
                ret

; connect a led to PORTC5, with the longer delay set above ie: ldi r27, 0x00  ;  zero so long enough delay to see a LED

;        generic delay so have time to see LEDs flash.  If you don't know how to connect a LED yet,Joe Pardue's
;“Aduino Workshop” is a good place to start (lots more than just connecting LEDs).

        delay_05:
                push r22
                push r24
                push r25
                ldi r22, 100      ;
        outer_loop:
                ldi r24, low(3037)
                ldi r25, high(3037)
        delay_loop:
                adiw r25:r24,1
```

```
                    brne delay_loop
                    nop
                    dec r22
                    brne outer_loop
                    pop r25
                    pop r24
                    pop r22
                    ret


;ISR   interupt service routine

EXT_INT0:
          reti
EXT_INT1:
PCINT0L:
PCINT1L:
PCINT2L:
WDT:
TIM2_COMPA:
TIM2_COMPB:
TIM2_OVF:
TIM1_CAPT:
TIM1_COMPA:
TIM1_COMPB:
TIM1_OVF:
          push r16
          in r16, SREG
          push r16
          clr r16
          sts TCCR1B,r16            ; off the timer
          cbi PORTC, 5              ;off the ping
          ldi r16, 2                ; go to state2
          pop r16
          out SREG, r16
          pop r16
          reti
TIM0_COMPA:
TIM0_COMPB:
TIM0_OVF:
SPI_STC:
USART_RXC:
USART_UDRE:
USART_TXC:
ADCR:
EE_RDY:
ANA_COMP:
TWI:
SPM_RDY:
          reti
```

Now The trigger is setup (remember to change the timer setting to 0xFF24).

After the trigger fires, the program is in state 2 waiting for the interrupt from the echo.

So during state 2, the main program is running doing other tasks.

Since we set up the interrupts, now we need the external int0 routines.

The following routine is called when any state change on PORTD2. It expects to be in state 2, waiting for the start of the echo, state3, waiting for the end of the echo, or state 5, a TOV interrupt iccurred first.

```
;ISR

EXT_INT0:
        push r16
        in r16, SREG
        push r16
        lds r16, GPIOR1             ; get state
        cpi r16, 2
        brne ic5
        clr r16                     ;state 2 must be the start of the echo
        sts TCNT1H, r16             ;high first ; interrupts are automatically turned off
        sts TCNT1L, r16
        ldi r16, 0b0000010          ; start clk, /8       here us the slower clock
        sts TCCR1B, r16
        ldi r16, 3                   ; next state 3     Started the timer and back to the main program
        rjmp eioend;
ic5:
        cpi r16, 3                  ;got the interrupt but after the overflow interrupt
        breq ic6                     ;set to state 6 because of overflow
        ldi r16, 6
        rjmp eioend
ic6:
        clr r16                      ; got the end
        sts TCCR1B, r16             ;stop the clock has the time (don't want to process in interrupt routine)
        ldi r16, 7
eioend:
        sts GPIOR1, r16  ;save the state
        pop r16
        out SREG, r16
        pop r16
        reti
EXT_INT1:
PCINT0L:
```

Here is the completed TOV interrupt routine.

Here there are two possibilities, entering in state1, waiting for the trigger pulse to time out
or some other error, most likely the timer timed out waiting for the echo pulse.

```
TIM1_OVF:
        push r16
        in r16, SREG
        push r16
        lds r16,GPIOR1               ; get state
        cpi r16,1
        brne tv2
        clr r16
        sts TCCR1B,r16          ; turn off the timer
        cbi PORTC, 5            ;trigger off
        ldi r16, 2              ;state 2
        rjmp t1oend
tv2:                            ;   error tov, distance too long
        clr r16
        sts TCCR1B,r16          ; turn off the timer
        ldi r16, 5

t1oend:
        sts GPIOR1,r16   ; save the state
        pop r16
        out SREG, r16
        pop r16
        reti
TIM0_COMPA:
```

Now let's look at the main loop of the final code:

```
loop:
                rcall quik                  ;routine to send state number to terminal
                lds r16, GPIOR1             ; get the state
                cpi r16, 0
                brne loop8
                rcall setst                 ; go start the timer
                rjmp loop
loop8:
                lds r16, GPIOR1
                cpi r16,7
                brne loop9
                rcall gotit     ;got a ping ;state 7 so send the timer results to the terminal
loop9:
                cpi r16, 5
                brne loop10
                rcall quik
                clr r16                     ;  error state
                sts GPIOR1, r16   ;
                rjmp loop
loop10:
                cpi r16,6
                brne loop
                rcall quik
                clr r16                     ; another error
                sts GPIOR1, r16
                rjmp loop

setst:          ; routine to set the trigger
                push r27
                push r26
                ldi r27, 0xFF               ;set the time for the counter for the trigger
                ldi r26, 0x24
                cli                         ;turn off interrupts
                sts TCNT1H,r27               ; high byte  set the time
                sts TCNT1L,r26               ; low
                sei                         ; interrupts back on
                ldi r16, 0b00000001
                sts TCCR1B, r16              ; start the timer with clk 1
                sts GPIOR1, R16              ; set state 1
                sbi PORTC,5             ;set the trigger high
                pop r26
                pop r27
                ret

gotit:          ;convert the time and send to the terminal
                push r19
                push r20
                cli
                lds r19, TCNT1L  ;  LOW FIRST
                lds r20, TCNT1H
                sei
                rcall getasc

                rcall send
                clr r16
                sts GPIOR1, r16  ; back to case 0
                rcall delay_05

                pop r20
                pop r19
                ret

quik:           ;used to send the state number to the terminal
                push r20
                push r19
                ldi r20,0
                lds r19,GPIOR1
                rcall getasc
                rcall send
                rcall delay_05
                pop r19
                pop r20
                ret
```

Thats the code.

Depending in the application yiu may want to convert the times to cm or in. Rather than doing the math. I measured out some distances and divided the number of ticks by the measured distances.
I did about 40 measurements, from 3cm to 290 cm. I found that I had subtract a small value, 22 ticks and use a conversion number of 115.6. The accuracy was better that .5% over the complete range.

I hope you found this of use.