**Working Principle of Flex Sensor(Flex Sensor 2.2"):**

A FLEX sensor is a transducer (a device that converts variations in a physical quantity, such as pressure or brightness, into an electrical signal, or vice versa) whose resistance is changed if we change its shape.



Fig. Flex Sensor

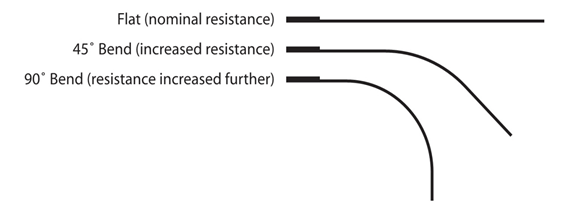


Fig. Flex Sensor’s resistance status

This sensor can sense the changes in linearity. So, when we bend this sensor, it’s resistance increases.

We use this sensor by converting its change in resistance to change in voltage. For this purpose, we need a voltage divider circuit.

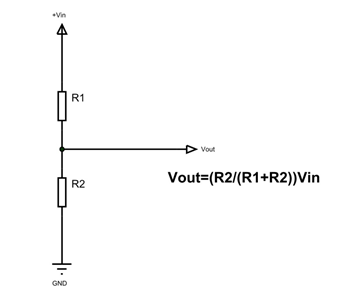


Fig. Voltage divider ckt for using Flex sensor

In the voltage divider circuit, we take a fixed resistance ( R1 ) and a Flex sensor ( R2 ). We take input from the midpoint and feed it to one of the ADC pins (PORT A in ATMEGA32). In the circuit, when the resistance R2 changes, i.e. we make change in the shape of the Flex sensor, the Vout changes linearly with it. So, in that sense, the voltage changes with the linearity of the Flex sensor.

Now, if we take one resistor and a Flex sensor, and form a divider circuit so that for Vin input voltage we get Vout in the output point (i.e. the middle point of the resistors), then in code we need to multiply the measured voltage with (Vin / Vout ) to get the original voltage.

In ADC, we’ll use the mode where AVCC = 5V is used as reference voltage, and so, we’ll need to adjust the whole voltage divider circuit in a way so that, we never need to measure more than 5V (i.e. configure the divider circuit, so that, Vout<=5V). Result ofADC of ATMEGA32 has 10 bits.

Now, to use the ADC, we need to setup two registers: ADMUX and ADCSRA.

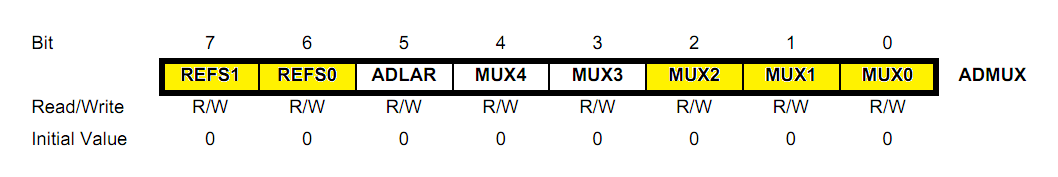


Fig. ADMUX register

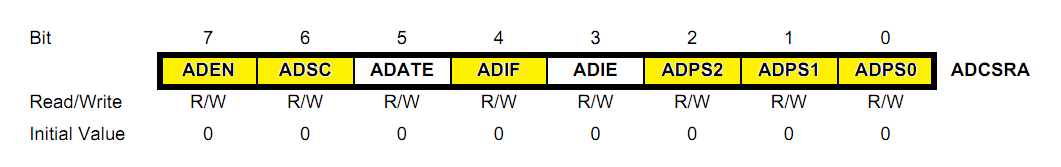


Fig. ADCSRA register

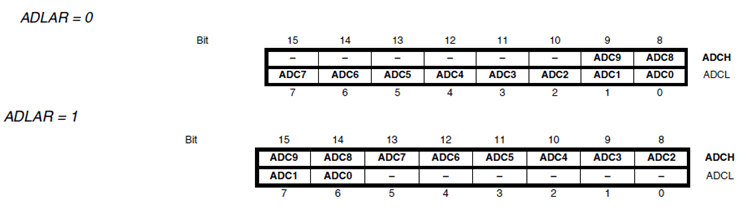


Fig. ADLAR’s value and the ADC result state

In ADMUX, we select the reference voltage type in the 7th and 6th bit, we’ll need to put 01 in those two as we’re using the fixed 5V reference. The ADLAR defines how the result will be adjusted in the result registers: ADCH and ADCL. If ADLAR=1, ADCH holds the upper 8 bits and ADCL holds the lower 2 bits, we need to do bit-masking to extract the result. If ADLAR=0, the vice-versa happens.

The last three bits of ADMUX selects the PIN we’re going to use out of the 8 available ADC pins, putting 000 selects PORTA0, putting 001 select PORTA1 and so on.

The first bit of ADCSRA, ADEN indicates if we’re using the ADC at all, i.e. putting 1 in this, enables the ADC feature of ATMEGA32. ADSC bit indicates the start of a conversion, and it remains high (i.e. 1), until the conversion is finished. ADATE, ADIF, ADIE indicates the features of auto-trigger enable, ADCinterrupt flag and ADC interrupt enable. The last three bits of ADCSRA works as Prescaler Select bits, these selects the clock cycle for the ADC unit. Different value in these bits, selects one of the pre-defined division factors. Putting 000 in these, selects ADC’s clock cycle as half of the ATMEGA32’s clock cycle, putting 010 select 1/4th and so on.

After configuration, we need to read the output from ADCL and ADCH, build the input voltage to this, by multiplying it with stepsize (VREF/210) .

Using some threshold value for this input, we can take different actions.

**Working principle for Sonar Sensor (Ultrasonic Sensor HC-SR04):**

The sensor uses ECHO technique to measure distance. ECHO is the thing we get when sound reflect back after striking a surface. Sound can’t go through solid surface. It gets reflected when a sound vibration hits a solid wall like surface. This is called ECHO.

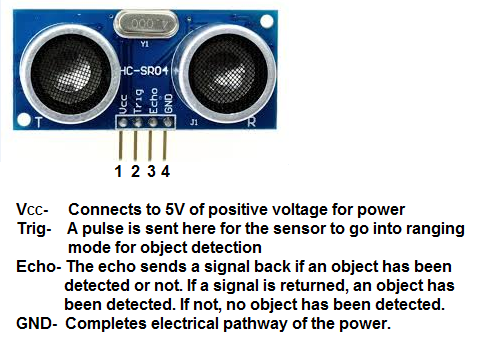


Fig. Sonar sensor HC-SR04

Ultrasonic sensor, HC-SR04 uses this technique to its advantage. It generates an output signal proportional to the distance based on the echo it receives. It has a trigger input, when we trigger it it generates a sound vibration in ultrasonic range. Then it waits for the sound vibration to return. Now Based on the parameters, i.e. velocity of sound and time taken for the echo to return, it provides output pulse proportional to distance.

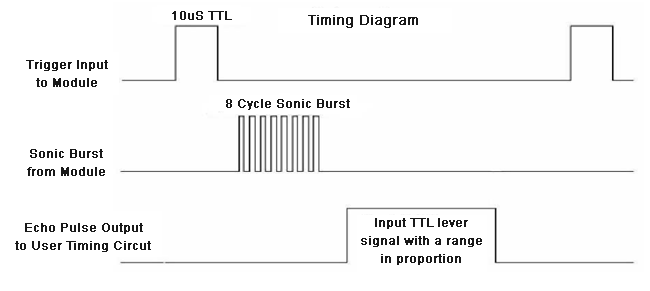


Fig. Sonar Sensor work flow

As we show in the figure above, before we take input from the sensor, we need to initiate the sensor for measuring distance. We need to put a HIGH logic in the Trig pin for more than 10 micro-second. A sound vibration is sent by sensor after that and the sensor waits for the echo, upon receiving the echo it generates a signal at the Echo pin whose width is proportional to the distance between the source and the obstacle.

The distance is, distance = width of the pulse output in micro-seconds/58 cm or

Width of the pulse output in micro-seconds/148 inch

The Pin-1 is given a voltage of +5V, which acts as the Vcc of the Sonar sensor, Pin-4 is shorted with the ground. We give input in the Trigger pin (Pin -2) and take input from Echo pin (Pin-3).

The ECHO pin is connected to INT0 (interrupt 0) pin of the ATMEGA32. This pin works as an external interrupt source.

So, we’ll first trigger the Trig-pin of the sonar for more than 10 micro-seconds. Once the ECHO pin goes high, we have an external interrupt and now we start a counter in the ISR we write, once ECHO goes low, we stop the counter. After this whole process, we have the width of the echo in the counter. We’ll need to do some more calculations to get the result.

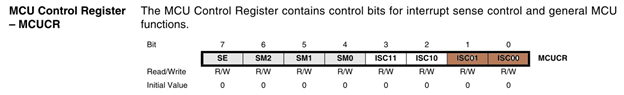


Fig. MCUCR register

We’re going to use INT0, so we need to use the rightmost two bits to specify how we want to trigger the interrupt. We need to put 01 in the registers correspondingly (from left), as we’re going to start a counter when the pin goes high and stop the counter when it goes low, so, any logical change on INT0 need to generate an interrupt request, which is exactly what ‘01’ mode stands for.

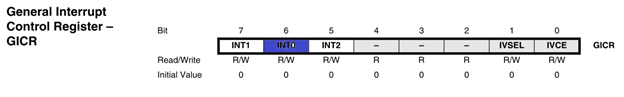


Fig. GICR register

We need to put 1 in the 6th bit of GICR to let the micro-controller know that we’re using the INT0.

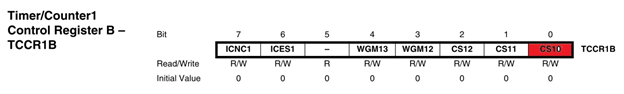


Fig. TCCR1B register

We need to put 1 in the left most bit of TCCR1B register to enable the counter and 0 to disable it. The count value after counting is stored in the 16bit TCNT1 register.

Code Sample:

1. //global variables
2. static volatile int pulse = 0;
3. static volatile int isHigh = 0; //indicates whether the ECHO is High
5. //do additional setups
7. while(1)
9. {
10. PORTD|=(1<<PIND0); //assuming we put the Trigger pin in PORTD0
12. \_delay\_us(20);///triggering the sensor for 20 micro-second
14. PORTD &=~(1<<PIND0); //reset the trigger
16. int distance = pulse/58;//getting the distance in cm
17. }
19. //ISR FOR THE INT0
20. ISR(INT0\_vect)
21. {
22. if (isHigh==1)
23. {
24. TCCR1B=0; //disable counter
25. pulse=TCNT1;//take the input in global variable
26. TCNT1=0; //clear the input
27. isHigh=0; //reset state
28. }
29. if (isHigh==0)
30. {
31. TCCR1B|=(1<<CS10); //enable counter
32. isHigh=1; //set the state to high
33. }
34. }