Project Name: Space Invaders

Team Members:

1. Nurul Muttakin [**13050009**] --- **01945546847**
2. S.Mahmudul Hasan (Numan) [**1305043**] --- **01521200014**, **01531508733**

Youtube Link:
https://www.youtube.com/watch?v=uVNDAjQfbOU

Features Implemented:
1. Basic game animation with Bi-color LED matrix
2. Player movement using Sonar sensor
3. Shooting bullets using Flex sensor

Basic Description:
In this game we've some enemies and a player. The player shoots eliminate the enemy mother ships. Once both the ships are destroyed the game is over and score is shown.
There are three types of enemies – Type-A, B and Mother ship. These looks like below--

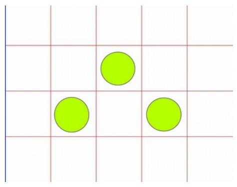


fig. Enemy A

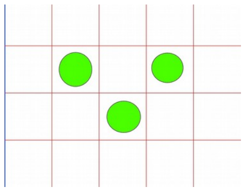


fig. Enemy B

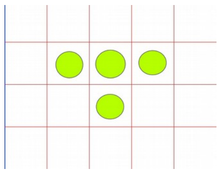


fig. Mother ship

Now, type A and B enemies comes down from top to bottom while keeping a left-right-left…. Motion. Mother ships stays on the very top of the game board and keeps moving from left-right and right-left.

The player needs to kill both the mother ships to win, regardless of killing all other enemies (A & B).

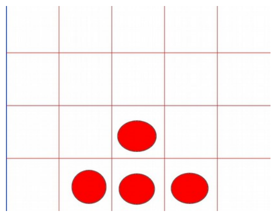


fig. Player

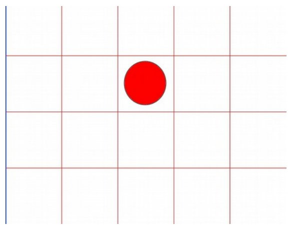


fig. Bullet

When the game is finished (either the player dies when the enemies collide with it, or the player kills both the mother ships), score is shown based on the number of kills.

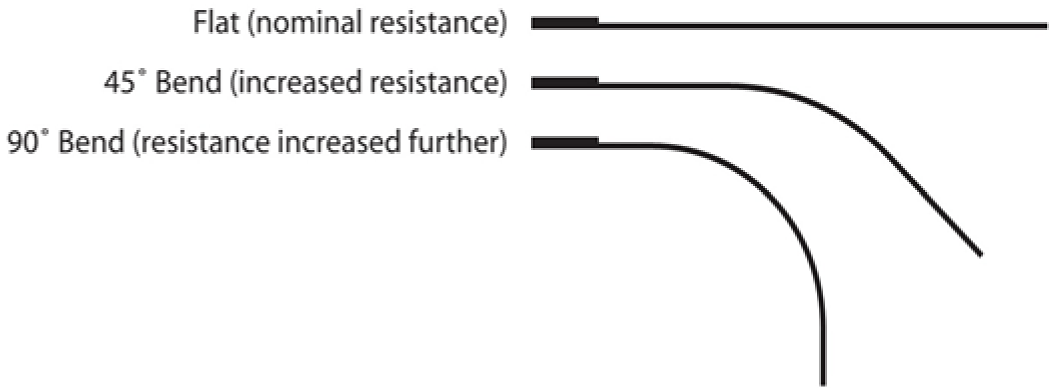
<u>Basic Working Principle:</u>

Block diagram, **Circuit diagram** and **Flow chart** are **added in separate files inside the zip folder** submitted.

<u>Interfacing Sonar and Flex sensors:</u>

<u>Working Principle of Flex Sensor(Flex Sensor 2.2"):</u>
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


Flat (nominal resistance)
45° Bend (increased resistance)
90° Bend (resistance increased further)
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.

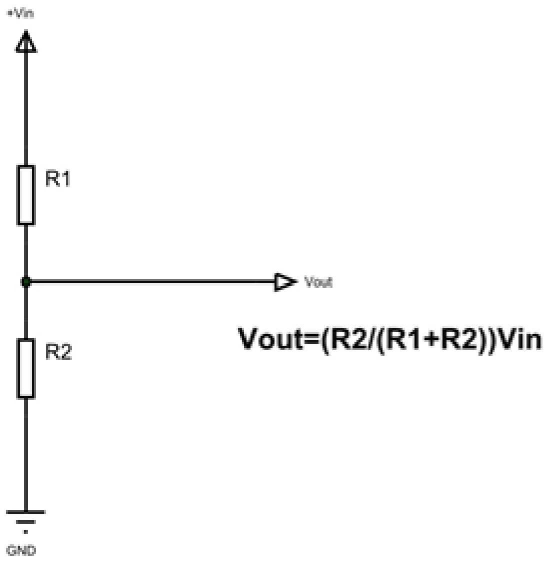
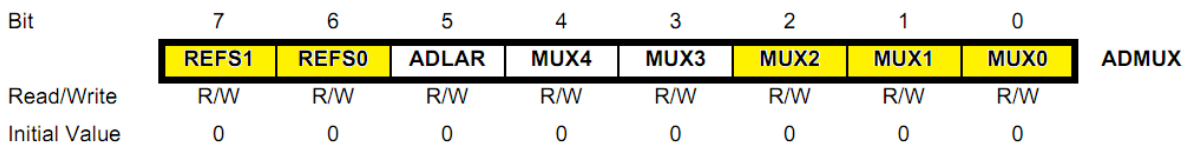


Fig. Voltage divider ckt for using Flex sensor

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 V_{out} changes linearly with it. So, in that sense, the voltage changes with the linearity of the Flex sensor.

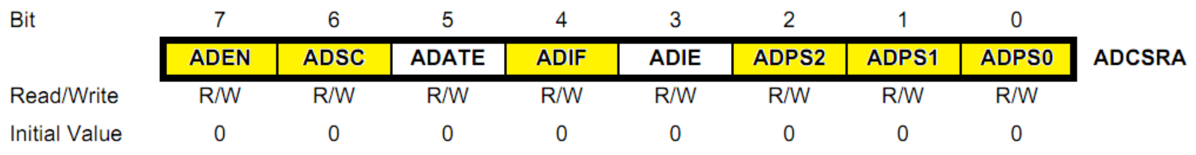
Now, if we take one resistor and a Flex sensor, and form a divider circuit so that for V_{in} input voltage we get V_{out} in the output point (i.e. the middle point of the resistors), then in code we need to multiply the measured voltage with (V_{in} / V_{out}) to get the original voltage.In ADC, we'll 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, V_{out}<=5V). Result ofADC of ATMEGA32 has 10 bits.

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



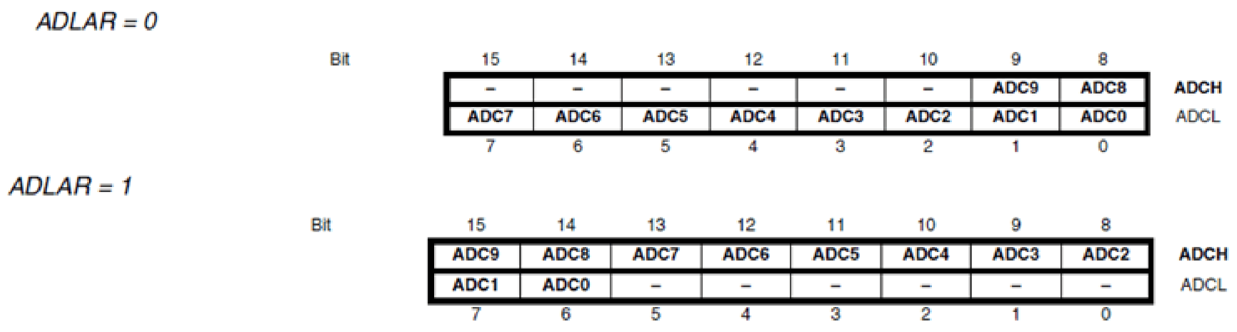
Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. ADMUX register



Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. ADCSRA register



ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

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.

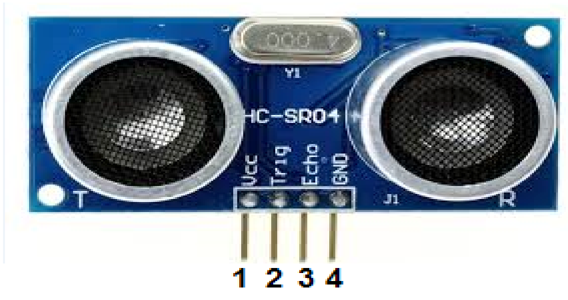
The first bit of ADCSRA, ADEN indicates if we’re using the ADC at all, i.e. putting 1 in this, enables the ADC feature of ATMEGA32. ADSC bit indicates the start of a conversion, and it remains high (i.e. 1), until the conversion is finished. ADATE, ADIF, ADIE indicates the features of auto-trigger enable, 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 ($V_{REF}/2^{10}$) .

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.



Vcc- **Connects to 5V of positive voltage for power**
Trig- **A pulse is sent here for the sensor to go into ranging mode for object detection**
Echo- **The echo sends a signal back if an object has been detected or not. If a signal is returned, an object has been detected. If not, no object has been detected.**
GND- **Completes electrical pathway of the power.**

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.

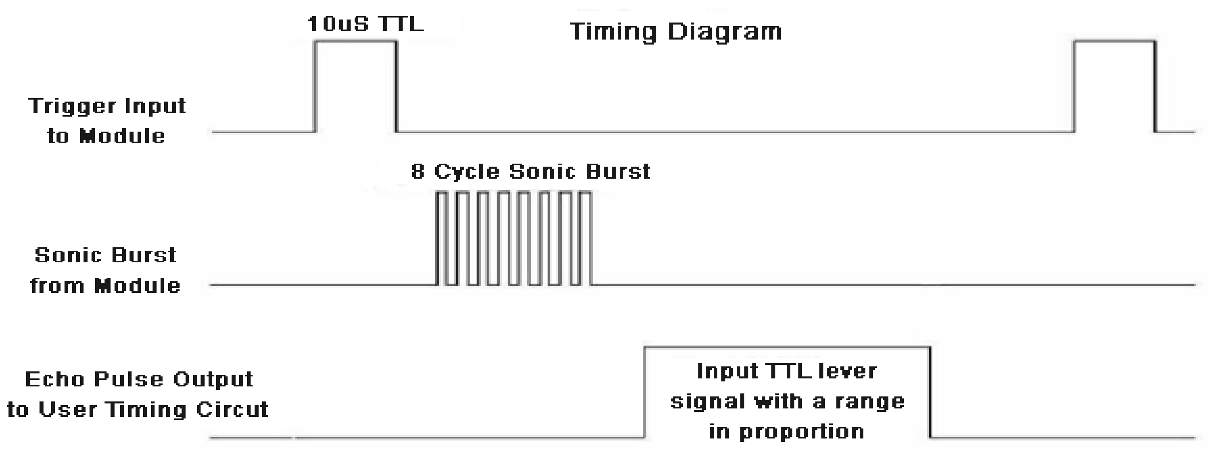

10uS TTL
Timing Diagram
Trigger Input to Module
8 Cycle Sonic Burst
Sonic Burst from Module
Echo Pulse Output to User Timing Circut
Input TTL lever signal with a range in proportion

Fig. Sonar Sensor work flow

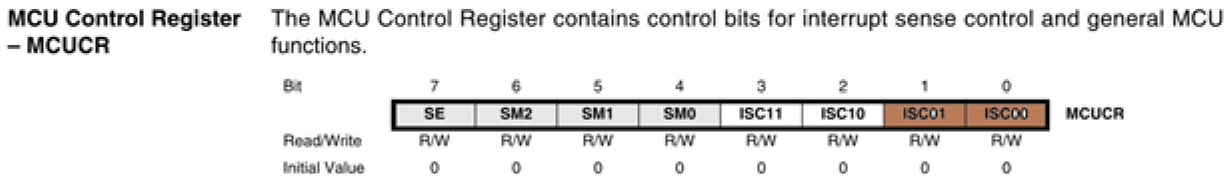
As we show in the figure above, before we take input from the sensor, we need to initiate the sensor for measuring distance. We need to put a HIGH logic in the Trig pin for more than 10 micro-second. A sound vibration is sent by sensor after that and the sensor waits for the echo, upon receiving the 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 V_{cc} 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.

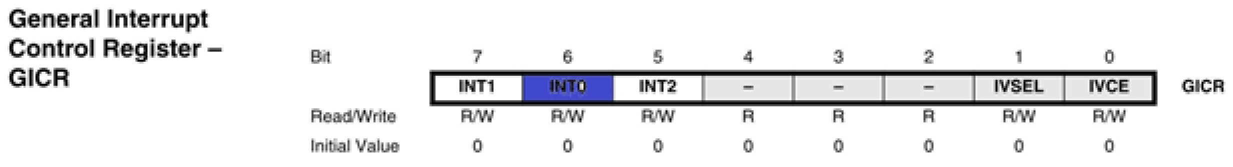


MCU Control Register – MCUCR The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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.

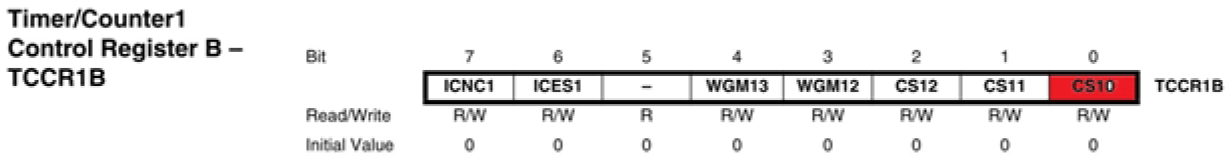


General Interrupt Control Register – GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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.



Timer/Counter1 Control Register B – TCCR1B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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:

```
//global variables
static volatile int pulse = 0;
static volatile int isHigh = 0; //indicates whether the ECHO is High

//do additional setups

while(1)

{
        PORTD|=(1<<PIND0); //assuming we put the Trigger pin in PORTD0

        _delay_us(20);///triggering the sensor for 20 micro-second

        PORTD &=~(1<<PIND0); //reset the trigger

        int distance = pulse/58;//getting the distance  in cm
}

//ISR FOR THE INT0
ISR(INT0_vect)
{
  if (isHigh==1)
  {
      TCCR1B=0; //disable counter
      pulse=TCNT1;//take the input in global variable
      TCNT1=0;      //clear the input
      isHigh=0;     //reset state
  }
  if (isHigh==0)
  {
      TCCR1B|=(1<<CS10); //enable counter
      isHigh=1;                 //set the state to high
  }
}
```

<u>**Working Principle of Dot Matrix Module**</u>:

We are using four 8x8 display dot matrix .We want a resolution of 16x16 dot matrix using these four dot matrix . We connected row so that it gives the resolution of 16 row by wiring and 16 column by wiring.

Now to connect these 16 rows and 16 columns we need 32 pins. So to reduce number of pins we used a 4 to 16 decoder (74HC154en) .So 16 columns will be connected through the decoder sixteen output pin. 16 pins of rows will be connected to the PORTD and PORTB. Input of the rows will be 5v but input of the column will be 0v to enlighten a specific point. As we can enlighten only one point at time. So we must have to use multiplexing to show any structure. Multiplexing is the process of showing only one point at a time but so fastly that our eyes can't differentiate the time interval so we will see this like a complete view. So using multiplexing we can do our task.

Problems and challenges we faced and how we overcame it:

1. We needed to interface four 8x8 LED dot matrix for our project. One Bi-color LED matrix usually takes 24 pins of the micro-controller out of the 32 available pins. In this way, we can never interface 4 8x8 Bi-color LED matrix with only 1 micro-controller. So, we used decoders to lessen the number of pins needed. We needed 2 decoders to interface 4 Bi-color LED matrix and it took in total 20 pins of the micro-controller.

2. Rendering simple animation – Normally we use one unsigned character for a row of 8 columns so, we needed 2 unsigned characters for a 16 column row. We couldn't make it work in this way, so we used 16 integers for 16 columns and it worked.

3. Power problem – we have an avr loader that was giving 3V although it was supposed to give 5V, so our code didn't seem to work. Later we used external 9V battery and a 9V-to-5V converter and the problem resolved.

4. Flex interfacing – while interfacing Flex sensor, using the above mentioned 3V avr loader, it didn't work, as we didn't give 5V but was calculating using 5V in the code, this problem was resolved when we used the 9V external battery.

5. Code – while coding we faced some common bugs and fixed them after looking for them for couple of hours.

6. Sonar interfacing – we interfaced sonar sensor rather easily. As we connected it after we had fixed the above mentioned voltage problem, it worked perfectly.

7. Bi-color interfacing – to make the project easier, we initially used single color build the whole project. After the whole project was working as it should, we interfaced the other one, but as the wires got too much tangled, it needed some time. The code part was rather cleaner and needed less time than we anticipated.