**<<The project>>**

**Abstract:**

\*No one can deny that revolution of artificial intelligence and it's effect in our life.

So we decide to use it to convert car from just object to intelligent object that can treat with you and represent it's status.

\*sometimes we need to check our car status. So our project helps you to know your car status by checking car profile.

\*in our system car send it's status such as ( speed, temperature, humidity, driver, place and oil level ) to it's profile. These information help car to represent itself and help any person to treat with it.

\*we added finger print system to help the owner to collect information about driver by sending private message to the owner.

\*The complete system model of our implementation system is divided as main modules :

1- Vehicle creates profile.

2- Vehicle checks status.

3- Vehicle send status to it's profile and send private message to the owner.

4- Vehicle treats with driver ( future step ).

Microcontroller At mega 32A



**Microcontroller At mega 32A**

Features :

Advanced RISC Architecture

.up to 16 MBIS throughput at 16MHZ

` .on chip 2-cycle multiplier

.fully static opoeration

Peripheral features:

.2 8bit timer/counters

.one 16bit timer/counter

.Real time counter with separate oscillator

.4PWM channels

.8 channel ,10-bit ADC

.programmable serial USART

.master/slave SPI serial interface

Special Microcontroller feature

.External and internal interrupt sources

.Six sleep modes(IDL, ADC noise reduction, power-save, power-Down, standby and extended Standby)

I/O and packages

.32 programmable I/O lines

.40 PIN PDIP, 44 lead TQFP and 44-pad QFN/MLF

Operating voltage:

.2.7 volt – 5.5 volt for ATMEGA 32l

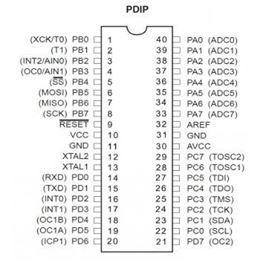
.4.5 volt- 5.5volt for ATMEGA 32

**Power consumption at 1 MHZ , 3volt 23 °C**

   .Active 1,1 mA

.IDLE mode .35mA

.Power Down mode < 1µA



An Ultrasonic Range Finder Module uses ultrasonic waves (inaudible to humans) to measure distance. These module consist of an Ultrasonic Transmitter (Tx) that emits the ultrasonic wave, the waves after striking any obstacle bounces back and reach the Ultrasonic Receiver (Rx). By measuring the time it take for the whole process to complete and using simple arithmetic we can measure the distance to the obstacle. The [**Ultrasonic Range Finder Modules**](http://shop.extremeelectronics.co.in/product_info.php?cPath=34&products_id=131) has a wide operating range of 1cm to 400cm with an accuracy of 1cm. These specifications makes it ideal for distance measurement application. **These can be used for :-**

* Contact less measurement of liquid level in tanks (even 4m deep tank!)
* Radars for robot.
* Obstacle sensing in Robotics.

* Speed check in roads.
  + Handheld units that can be pointed on vehicles to measure their speed.
  + Fixed unit installed in check booths that can click pictures of over speeding vehicles (Remember NFS Most Wanted?)

**The reasons for using ultrasonic wave are:-**

* The speed of Ultra Sonic waves is 343m/s ([Speed of Sound](http://en.wikipedia.org/wiki/Speed_of_sound)) which is **not too fast** for MCUs to measure accurately. Compare this with speed of electromagnetic waves (like light or radio waves) which is 30,00,00,000 m/s! So it takes only 20ns (Nano second) to go and bounce back from an obstacle which is 3m away! An AVR running at 16MIPS (maximum for most AVRs) takes 62ns to execute a single instruction.
* Ultrasonic waves travels more narrow, like a beam than normal sound wave. This property helps the sensor detect the obstacles that are exactly in line with it only. The sensors can be rotated with steppers or [servo motors](http://extremeelectronics.co.in/avr-tutorials/servo-motor-control-by-using-avr-atmega32-microcontroller/) to get a "image" of obstacle in the surrounding area (like a radar).
* Finally the waves don’t disturb any humans nearby!

1- Microcontroller makes the i/o line output. (By using the [DDRx Register](http://extremeelectronics.co.in/avr-tutorials/part-v-digital-io-in-avrs/) in AVR or [TRISx Register](http://extremeelectronics.co.in/microchip-pic-tutorials/general-purpose-digital-io-with-pic-microcontrollers/) in PIC)

1. the i/o line is made low (this may be the default state of i/o pin)
2. Wait for 10uS
3. Make the I/O line high.
4. Wait for 15uS
5. Make the i/o line low
6. Wait for 20uS
7. Now make it input (by using the [DDRx Register](http://extremeelectronics.co.in/avr-tutorials/part-v-digital-io-in-avrs/) in AVR)
8. Module will keep it low. Wait till it is low, as soon as it becomes high start the [**timer**](http://extremeelectronics.co.in/avr-tutorials/avr-timers-an-introduction/).
9. After that wait till it is high, as soon as it becomes low copy the [**timer**](http://extremeelectronics.co.in/avr-tutorials/avr-timers-an-introduction/) value and stop the timer.
10. Finally we have the time required for the wave to go hit the obstacle and come back to the module.

If the pulse width is in microseconds, the distance can be calculated by the following formula :-

* Distance in cm = Pulse width/58
* Distance in inches = Pulse width/148

**Ultrasonic Range Finder Interfacing Sample code for AVR**

To understand our code for Ultrasonic Range Finder Interface you need to have basic knowledge about timers in microcontroller. In short a timer is a register which increments it value at predefined intervals without the need of any help from CPU. One of the uses of timer is to measure the time accurately. Please note that modern MCUs have sophisticated timers with lots of configuration options and can be used for several purposes.

Here we will use TIMER1 of ATmega32 for counting the duration of pulse. The TIMER1 has 15 different waveform generation modes. We will use the default that is called NORMAL mode. The modes are selected using 4 bits called WGM13,WGM12,WGM11,WGM10. We need the NORMAL mode which is applied by setting all four bits to 0. WGM1 stands for **W**aveform **G**eneration **M**ode for timer **1**.

To setup the TIMER1 we need to setup its control registers properly. They are called

* TCCR1A – Timer Counter Control Register 1 A
* TCCR1B – Timer Counter Control Register 1 B

**TCCR1A – Timer Counter Control Register 1 A**

This register basically deals with the Output Compare Modes so they are used when generating PWM signal from timer. As we are using timer in NORMAL mode we set most Output Compare related bits to 0. This register also has the WGM11 and WGM10, as discussed earlier we need to set all WGM1x bits to 0. This results in all 8 bits in TCCR1A set to 0. The line below is taken from example code.

TCCR1A=0X00;

**TCCR1B – Timer Counter Control Register 1 B**

This register has bits related to Input Capture, WGM13,WGM12 and the Clock Select Bits (CS12,CS11,C10). We need to set the input capture related bits to 0 and the WGM1x bits to 0 too. The final thing is the CS1x bits. They are used to select a clock source for TIMER1 as per the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **CS12** | **CS11** | **CS10** | **Description** |
| 0 | 0 | 0 | No Clock Source (Timer/Counter Stopped) |
| 0 | 0 | 1 | clki/o/1 (No Prescaling) |
| 0 | 1 | 0 | clki/o/8 (No Prescaling) |
| 0 | 1 | 1 | clki/o/64 (No Prescaling) |
| 1 | 0 | 0 | clki/o/256 (No Prescaling) |
| 1 | 0 | 1 | clki/o/1024 (No Prescaling) |
| 1 | 1 | 0 | See data sheet page 110 |
| 1 | 1 | 1 | See data sheet page 110 |

We are running on CPU speed of 16MHz, so our I/O clock is 16MHz. We divide this by 8 to get a 2MHz clock for our timer. This means the timer increments its value in 0.5 microseconds. We can divide the value by two to get exact 1 microsecond time base. So the TCCR1B is configured by only setting up CS11, this is written in C as follows

TCCR1B= (1<<CS11);

After setting up the timer using the two control registers, we clear the counter by writing 0 to it

TCNT1=0x00; **//Init counter**

After that we wait for the falling edge and as soon as it is detected we put the timer value in a temporary variable called ***result***.

**//Falling edge found**

result=TCNT1;

**//Stop Timer**

TCCR1B=0x00;

It is highly recommended to go through the Chapter **16bit timer/counter1** in the datasheet of ATmega32. TCCR1A and TCCR1B is described in page 107 to 110 of the datasheet.

The code employ some error checking and preprocessor magic that may confuse you but they are necessary for

* Preventing the code to hang the system if uSonic Module has errors or Not connected. (Waiting forever for rising/falling edges). So we employ a time-out system.
* Allows you to easily change the PORT and Position where the sensor is connected.

