Embedded Systems Project

Automotive application 2: Facilitating parking maneuvers by measuring the distance remaining to an obstacle using a micro controller distance sensors

Author: Daniel-Ioan Giuriciu
Study year: 3
Group: 3.2
School year: 2017-2018

Specifications, characteristics and requirements:

- It is recommended to use the “Capture – Compare module of the micro controller in order to periodically generate events

- The distance sensor may be connected to a serial interface, such as CAN, if the designer so wishes.

- The distance measured shall be between values of 10 – 70, and 70 – 100 centimeters.

- The measurement will be accompanied by a sound alarm. The sounds frequency will rise as the distance to the obstacle decreases.

- The value of the measured distance shall be displayed on a led matrix, a seven segment display or an LCD.

- At least 2 distance sensors shall be used, and the application will have a practical prototype

Board Description:

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.
The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip.
Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

Board Specifications:

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Power:

The Arduino Uno can be powered via the USB connection or with an external power supply. The power
source is selected automatically.
External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.
The power pins are as follows:
- VIN. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts
from the USB connection or other regulated power source). You can supply voltage through this pin, or, if

supplying voltage via the power jack, access it through this pin.
- 5V.This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either
from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying
voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- GND. Ground pins.
- IOREF. This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A
properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable
voltage translators on the outputs for working with the 5V or 3.3V.

Input and Output:

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a
maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:
- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the
corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling
edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the
pin is LOW, it's off.
The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e.
1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:
- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.
There are a couple of other pins on the board:
- AREF. Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block
the one on the board.
See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

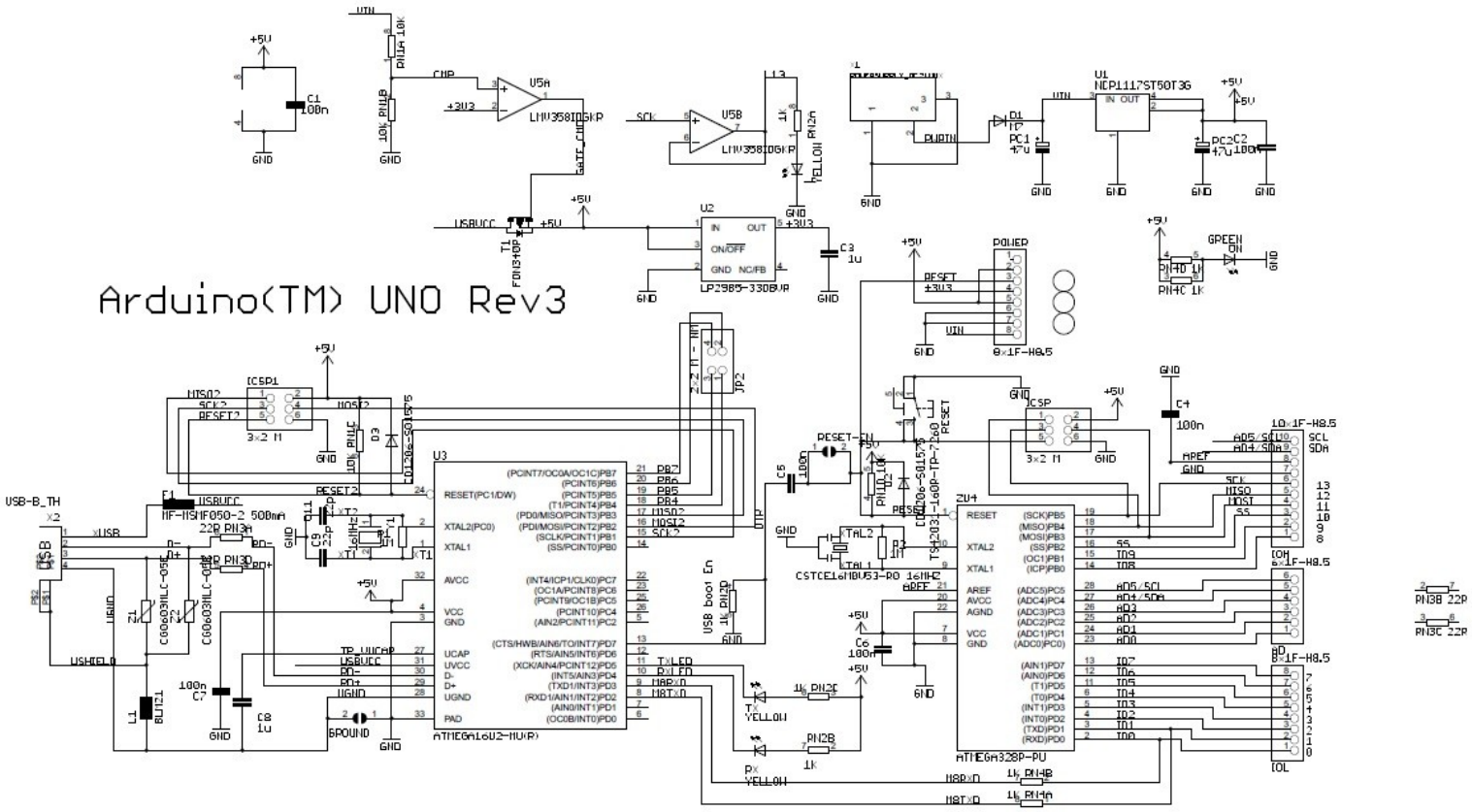
Communication:

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or
other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple
textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.
The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPIcommunication, use the SPI library.

Programming:

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials.
The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).
You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.
The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:
- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then
resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to
put into DFU mode.
You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to
load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Board Schema:

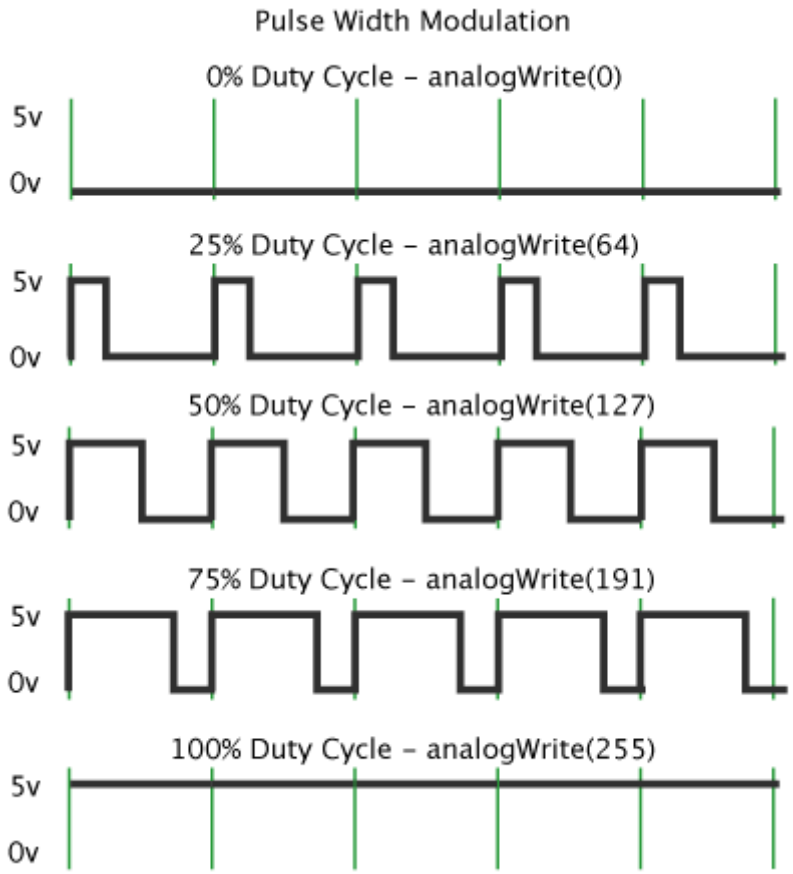


PWM:

The Fading example demonstrates the use of analog output (PWM) to fade an LED. It is available in the File->Sketchbook->Examples->Analog menu of the Arduino software.

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.



USB Module:

Used to both provide power and program the board. Based on FTDI FT232 circuit.

The FT232R is the latest device to be added to FTDI's range of USB UART interface Integrated Circuit Devices. The FT232R is a USB to serial UART interface with optional clock generator output, and the new FTDIChip-ID™ security dongle feature. In addition, asynchronous and synchronous bit bang interface modes are available. USB to serial designs using the FT232R have been further simplified by fully integrating the external EEPROM, clock circuit and USB resistors onto the device.

2 FT232R Block Diagram

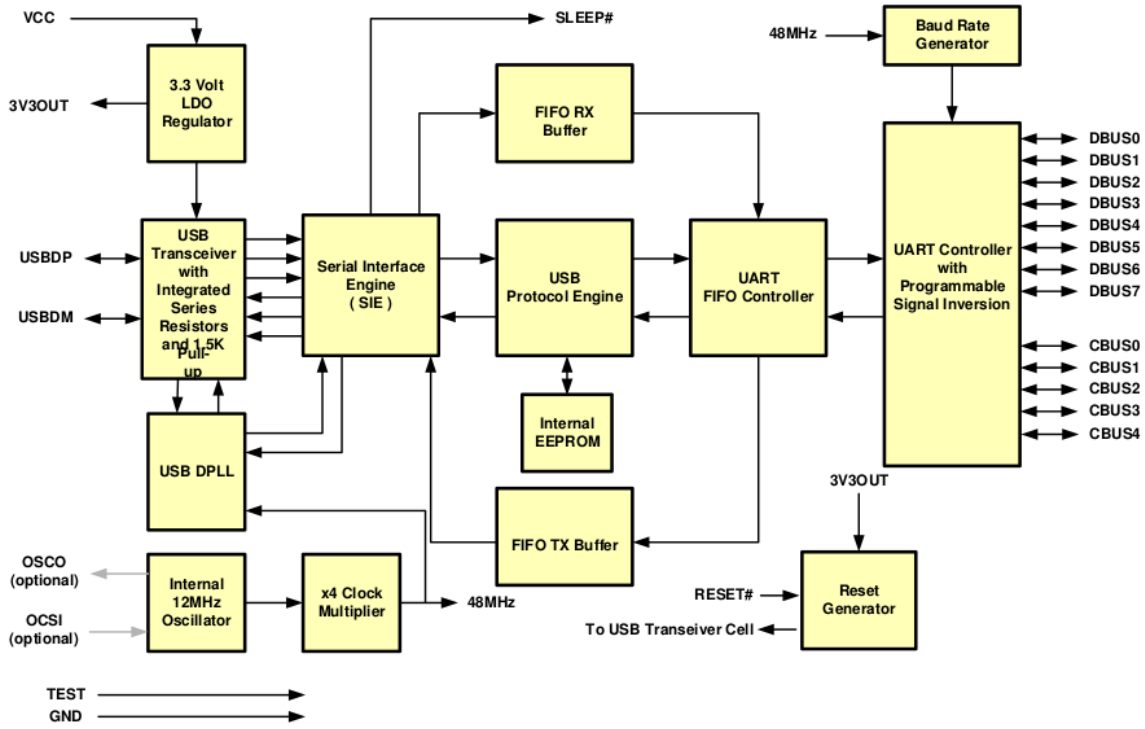


Figure 2.1 FT232R Block Diagram

Timers:

What is a timer?
A timer or to be more precise a timer / counter is a piece of hardware builtin the Arduino controller (other controllers have timer hardware, too). It is like a clock, and can be used to measure time events.
The timer can be programmed by some special registers. You can configure the prescaler for the timer, or the mode of operation and many other things.
The controller of the Arduino is the Atmel AVR ATmega168 or the ATmega328. These chips are pin compatible and only differ in the size of internal memory. Both have 3 timers, called timer0, timer1 and timer2. Timer0 and timer2 are 8bit timer, where timer1 is a 16bit timer. The most important difference between 8bit and 16bit timer is the timer resolution. 8bits means 256 values where 16bit means 65536 values for higher resolution.
The controller for the Arduino Mega series is the Atmel AVR ATmega1280 or the ATmega2560. Also identical only differs in memory size. These controllers have 6 timers. Timer 0, timer1 and timer2 are identical to the ATmega168/328. The timer3, timer4 and timer5 are all 16bit timers, similar to timer1.
All timers depends on the system clock of your Arduino system. Normally the system clock is 16MHz, but for the Arduino Pro 3,3V it is 8Mhz. So be careful when writing your own timer functions.
The timer hardware can be configured with some special timer registers. In the Arduino firmware all timers were configured to a 1kHz frequency and interrupts are gerally enabled.

Timer0:
Timer0 is a 8bit timer.
In the Arduino world timer0 is been used for the timer functions, like delay(), millis() and micros(). If you change timer0 registers, this may influence the Arduino timer function. So you should know what you are doing.

Timer1:
Timer1 is a 16bit timer.
In the Arduino world the Servo library uses timer1 on Arduino Uno (timer5 on Arduino Mega).

Timer2:
Timer2 is a 8bit timer like timer0.
In the Arduino work the tone() function uses timer2.

Timer3, Timer4, Timer5:
Timer 3,4,5 are only available on Arduino Mega boards. These timers are all 16bit timers.

Timer Register
You can change the Timer behaviour through the timer register. The most important timer registers are:
TCCRx - Timer/Counter Control Register. The prescaler can be configured here.

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x81)	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	

Interrupts:

Interrupts are useful for making things happen automatically in microcontroller programs, and can help solve timing problems. Good tasks for using an interrupt may include reading a rotary encoder, or monitoring user input.

If you wanted to insure that a program always caught the pulses from a rotary encoder, so that it never misses a pulse, it would make it very tricky to write a program to do anything else, because the program would need to constantly poll the sensor lines for the encoder, in order to catch pulses when they occurred. Other sensors have a similar interface dynamic too, such as trying to read a sound sensor that is trying to catch a click, or an infrared slot sensor (photo-interrupter) trying to catch a coin drop. In all of these situations, using an interrupt can free the microcontroller to get some other work done while not missing the input.

ISRs are special kinds of functions that have some unique limitations most other functions do not have. An ISR cannot have any parameters, and they shouldn't return anything.

Generally, an ISR should be as short and fast as possible. If your sketch uses multiple ISRs, only one can run at a time, other interrupts will be executed after the current one finishes in an order that depends on the priority they have. millis() relies on interrupts to count, so it will never increment inside an ISR. Since delay() requires interrupts to work, it will not work if called inside an ISR. micros() works initially, but will start behaving erratically after 1-2 ms. delayMicroseconds() does not use any counter, so it will work as normal.

Typically global variables are used to pass data between an ISR and the main program. To make sure variables shared between an ISR and the main program are updated correctly, declare them as `volatile`.

BOARD	DIGITAL PINS USABLE FOR INTERRUPTS
Uno, Nano, Mini, other 328-based	2, 3
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR1000 Rev.1	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Due	all digital pins
101	all digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with CHANGE)

Syntax

`attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);` (recommended)
`attachInterrupt(interrupt, ISR, mode);` (not recommended)
`attachInterrupt(pin, ISR, mode);` (not recommended Arduino Due, Zero, MKR1000, 101 only)

Parameters

`interrupt`: the number of the interrupt (`int`)
`pin`: the pin number *(Arduino Due, Zero, MKR1000 only)*
`ISR`: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.
`mode`: defines when the interrupt should be triggered. Four constants are predefined as valid values:

- **LOW** to trigger the interrupt whenever the pin is low,
- **CHANGE** to trigger the interrupt whenever the pin changes value
- **RISING** to trigger when the pin goes from low to high,
- **FALLING** for when the pin goes from high to low.
  The Due, Zero and MKR1000 boards allows also:
- **HIGH** to trigger the interrupt whenever the pin is high.

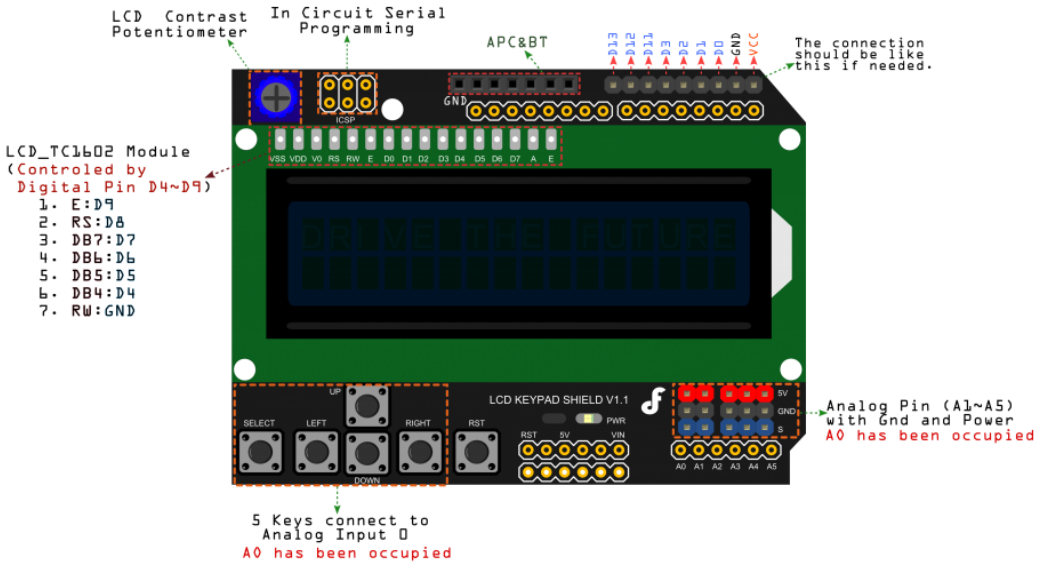
LCD Shield:

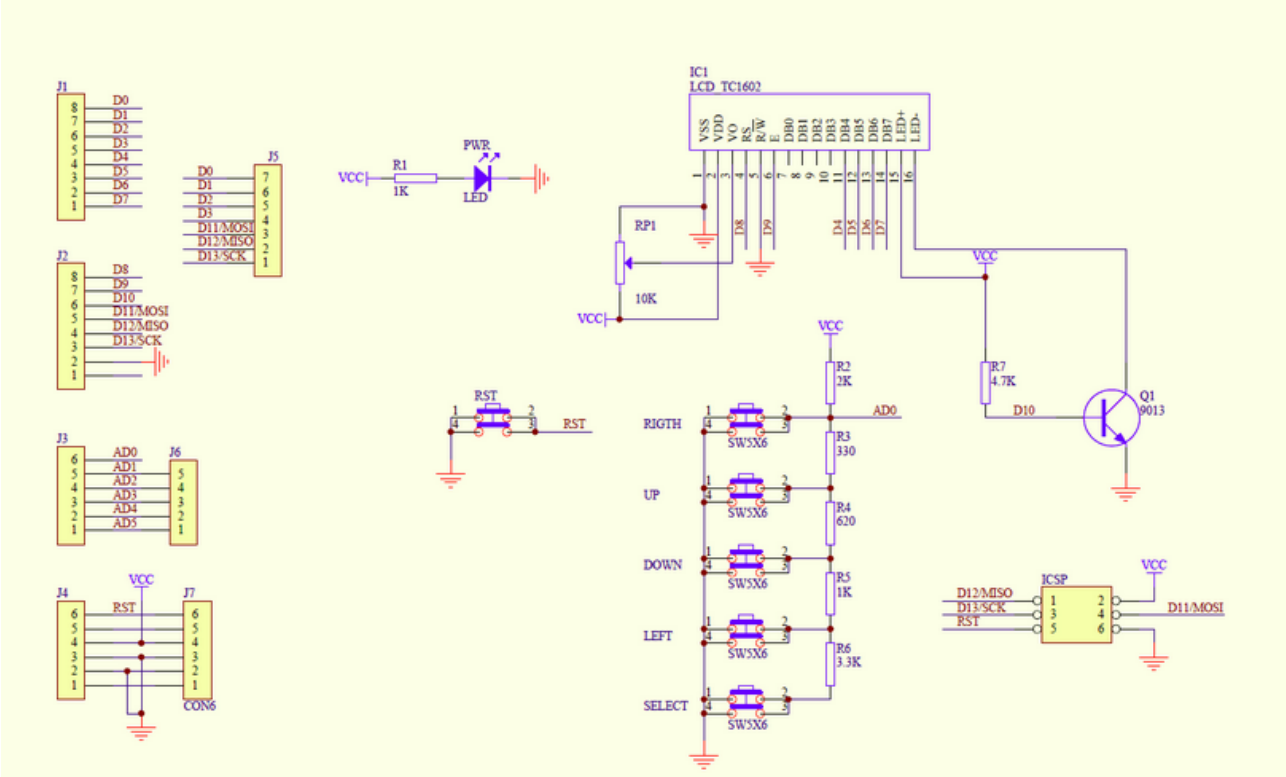
The **arduino *LCD Keypad shield*** is developed for **Arduino compatible boards**, to provide a user-friendly interface that allows users to go through the menu, make selections etc. It consists of a 1602 white character blue backlight LCD. The keypad consists of 5 keys — select, up, right, down and left. To save the digital IO pins, the keypad interface uses only one ADC channel. The key value is read through a 5 stage voltage divider.

Operating Voltage:5V

- 5 Push buttons to supply a custom menu control panel
- RST button for resetting arduino program
- Integrate a potentiometer for adjusting the backlight
- Expanded available I/O pins
- Expanded Analog Pinout with standard DFRobot configuration for fast sensor extension
- Dimension: 80 x 58 mm

Pinout





Function Explanation

LiquidCrystal(rs, enable, d4, d5, d6, d7)

Creates a variable of type LiquidCrystal. The display can be controlled using 4 or 8 data lines. If the former, omit the pin numbers for d0 to d3 and leave those lines unconnected. The RW pin can be tied to ground instead of connected to a pin on the Arduino; if so, omit it from this function's parameters. for example:

```
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
```

lcd.begin(cols, rows)

Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display. begin() needs to be called before any other LCD library commands.for example:

```
lcd.begin(16, 2);
```

lcd.setCursor(col,row)

Set the location at which subsequent text written to the LCD will be displayed. for example:

```
lcd.setCursor(0,0);
```

lcd.print(data)

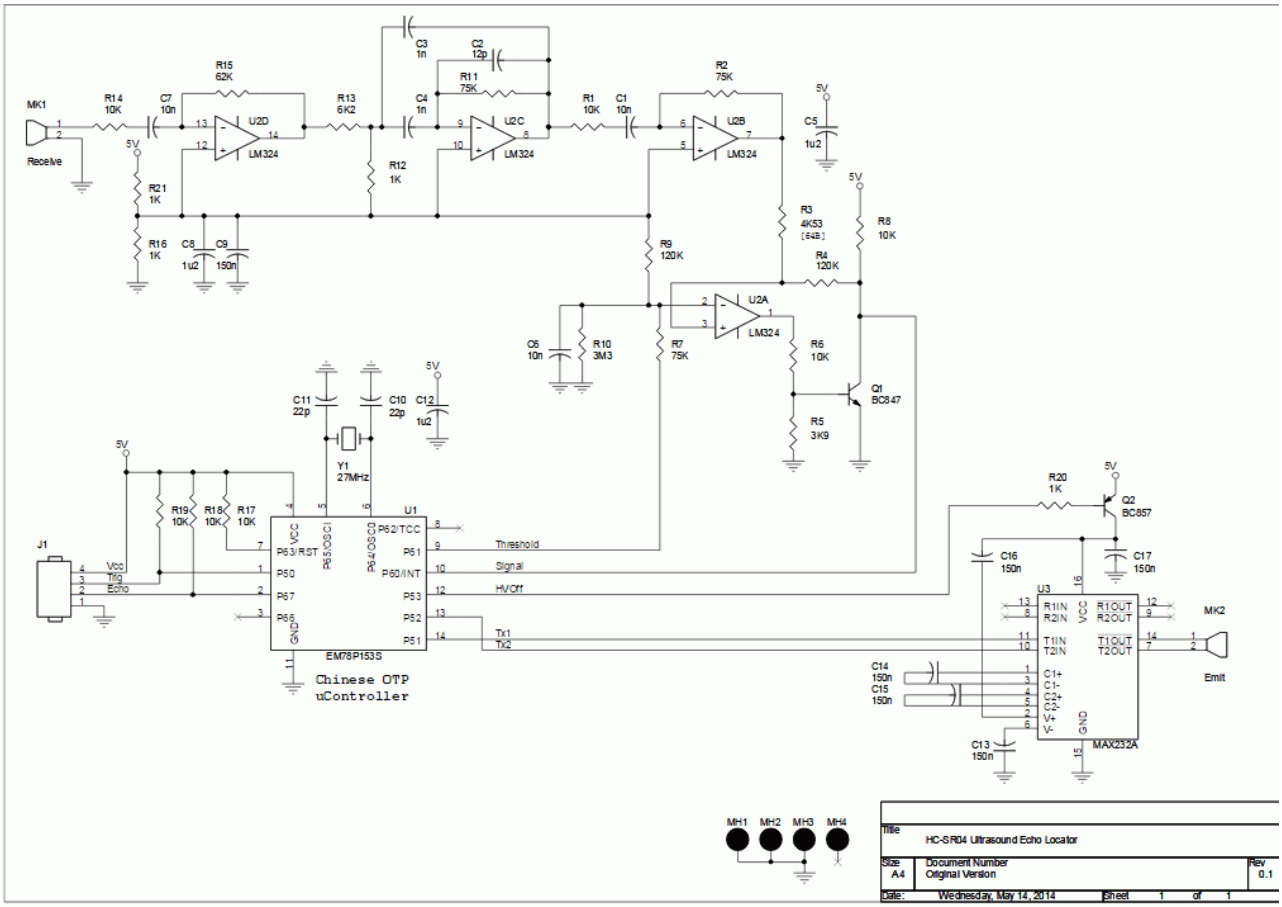
Prints text to the LCD.for example:

```
lcd.print("hello, world!");
```

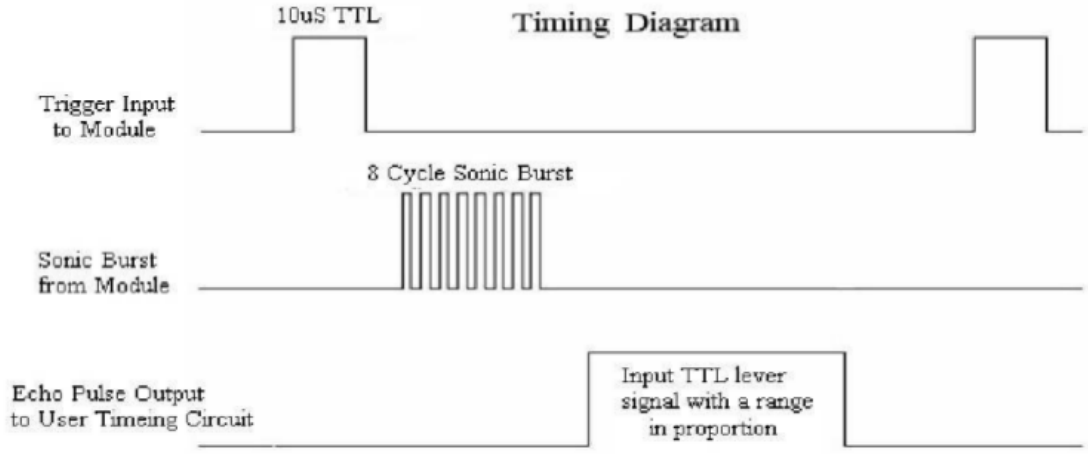
lcd.write(data)

Write a character to the LCD.

Distance Sensor HC-SR04:

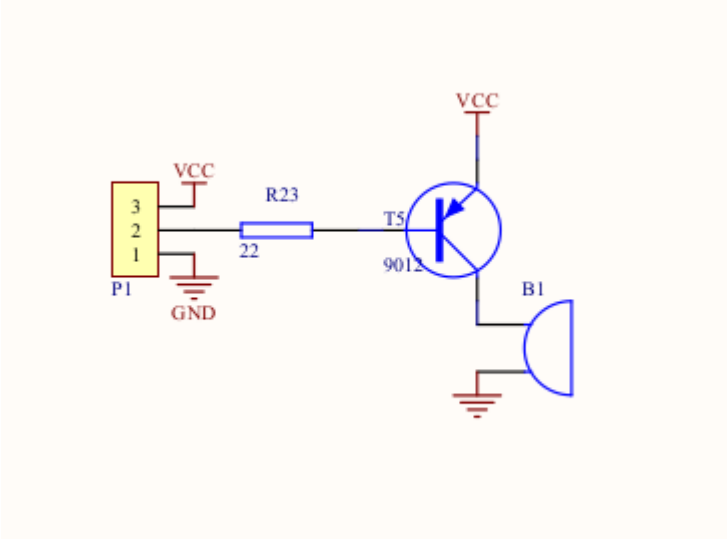


The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal.



YL-44 Buzzer:

This is a small buzzer module which operates around the audible 2 kHz frequency range. It is an active buzzer, which means that it produces sound by itself, without needing an external frequency generator.



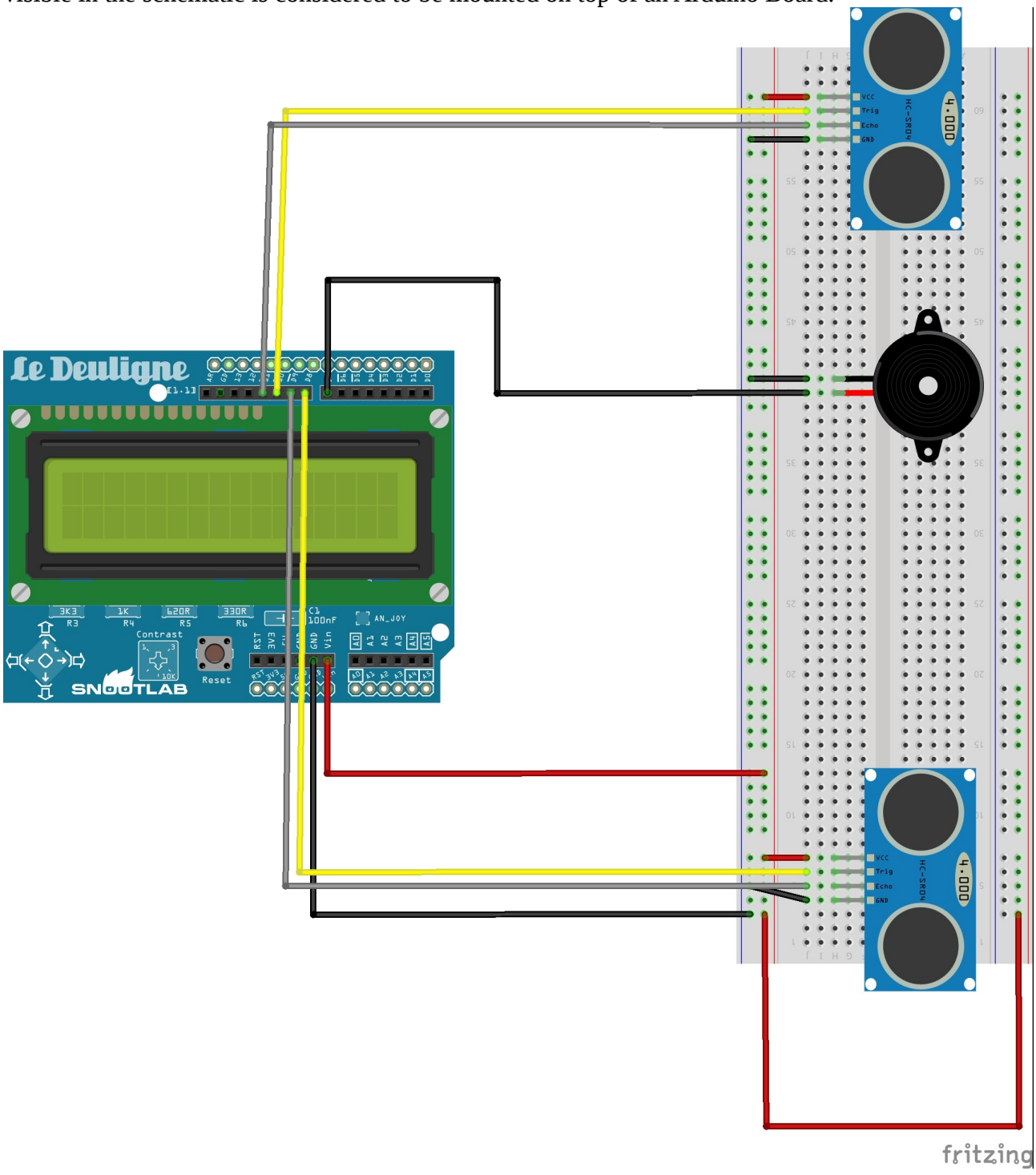
Code:

```
#include<LiquidCrystal.h>

LiquidCrystal lcd(7,6,5,4,3,2); //config params for LCD shield, see docs for details

//configuring pins for HC-SR04
const int trig1 = 10;
const int trig2 = 8;
const int echo1 = 11;
const int echo2 = 9;

const int buzzPin = 7; //buzzer pin
long duration;
int distanceCm1;
int distanceCm2;


void setup() //function that will execute upon powerup
{
  lcd.begin(16,2); // Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display
//setup pins
  pinMode(trig1, OUTPUT);
  pinMode(echo1, INPUT);
  pinMode(buzzPin, OUTPUT);
  pinMode(trig2, OUTPUT);
  pinMode(echo2, INPUT);
  Serial.begin(9600); //Used serial monitoring for debugging purposes
}


void loop()  //function that will execute on every controller cycle
{
  digitalWrite(buzzPin, HIGH); //stop buzzer pin
  digitalWrite(trig1, LOW); //make sure the HC-SR04 is not triggered by noise and trig pin is clear
  delayMicroseconds(2);
  digitalWrite(trig1, HIGH);  //generate ultrasound
  delayMicroseconds(10);
  digitalWrite(trig1, LOW); //stop ultrasound generation
  duration = pulseIn(echo1, HIGH); //read echo pin and store retrieved value in duration
  distanceCm1 = duration*0.034/2; //compute distance

  //redo same operation for second HC-SR04
  digitalWrite(trig2, LOW);
  delayMicroseconds(2);
```

```
  digitalWrite(trig2, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig2, LOW);
  duration = pulseIn(echo2, HIGH);
  distanceCm2 = duration*0.034/2;


  lcd.setCursor(0,0); // Sets the location at which subsequent text written to the LCD will be
displayed
  lcd.print("Distance: "); // Prints string "Distance" on the LCD
  lcd.print(distanceCm1); // Prints the distance value from the sensor
  lcd.print(" cm"); //Print unit measurements

  lcd.setCursor(0,1); //move down to the second row
  lcd.print("Distance: ");
  lcd.print(distanceCm2);
  lcd.print(" cm");

//Also print to serial monitor, debugging purposes
  Serial.print("Distance1 is: ");
  Serial.println(distanceCm1);

  Serial.print("Distance2 is: ");
  Serial.println(distanceCm2);

//buzzer activation logic

//distance is less than 30cm → critical, activate buzzer with maximum frequency
  if ((distanceCm1 < 30) && (distanceCm1 > 0) || (distanceCm2 < 30) && (distanceCm2 > 0))
  {
    digitalWrite(buzzPin, LOW);
  }
//distance is less than 70cm → dangerous, activate buzzer with mid-way frequency
  else if ((distanceCm1 < 70) && (distanceCm1 > 0) || (distanceCm2 < 70) && (distanceCm2 > 0))
  {

    digitalWrite(buzzPin, LOW);
    delay(2000); //add delay to alter frequency
    digitalWrite(buzzPin, HIGH);
  }

//distance is 1
  else if ((distanceCm1 < 100) && (distanceCm1 > 0) || (distanceCm2 < 100) && (distanceCm2 >
0))
  {

    digitalWrite(buzzPin, LOW);
    delay(1000);
    digitalWrite(buzzPin, HIGH);
  }
  delay(1000);
}
```

Hardware & connections:

Due to some parts being unavailable, and considering wiring becoming overly cluttered, the orientation of the HC-SR04 has been changed(both facing in the same direction, and the YL-44 buzzer has been replaced with a basic buzzer controlled by digital pin 7. Also, the LCD shield visible in the schematic is considered to be mounted on top of an Arduino Board.



Resources and references:

* https://www.arduino.cc/
* https://circuits.io/search/?q=arduino%20uno
* https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/
* https://www.wiley.com/en-us/Arduino+For+Dummies-p-9781118446379
* https://electronics.stackexchange.com/questions/224374/active-vs-passive-buzzer
* https://www.dfrobot.com/wiki/index.php/Arduino_LCD_KeyPad_Shield_(SKU:_DFR0009)
* http://www.thomasclausen.net/en/walking-through-the-1602-lcd-keypad-shield-for-arduino/
* See datasheets for YL-44 and HC-SR44