Green House Controller App

System Guide

Mitic, Vladimir

2015

# **Index**

Table of Contents

[**Index** 0](#_Toc412047984)

[**Introduction** 0](#_Toc412047985)

[Why Arduino? 0](#_Toc412047986)

[Components 0](#_Toc412047987)

[Arduino Uno R3 Board 0](#_Toc412047988)

[Add-on Shields (Ethernet Shield R3) 0](#_Toc412047989)

[Sensors 0](#_Toc412047990)

[Arduino Code and Libraries 0](#_Toc412047991)

[Website 0](#_Toc412047992)

[**Other Shields** 0](#_Toc412047993)

[WiFiShield 0](#_Toc412047994)

[**Sensors** 0](#_Toc412047995)

[Air Humidity and Temperature Sensor 0](#_Toc412047996)

[Real Time Clock (RTC) 0](#_Toc412047997)

[Waterproof Digital Sensors 0](#_Toc412047998)

[Ultra-Sonic Distance Sensor 0](#_Toc412047999)

[Light Sensor 0](#_Toc412048000)

[PH Sensor 0](#_Toc412048001)

[Tilt Switch 0](#_Toc412048002)

[**Outputs** 0](#_Toc412048003)

[LCD Screen 0](#_Toc412048004)

[**Hardware Interfaces** 0](#_Toc412048005)

[SPI 0](#_Toc412048006)

[I2C 0](#_Toc412048007)

[1Wire 0](#_Toc412048008)

[**Hardware Design** 0](#_Toc412048009)

[**Software Design** 0](#_Toc412048010)

[Web design 0](#_Toc412048011)

[C design 0](#_Toc412048012)

[Behind the scenes 0](#_Toc412048013)

[Ways to optimize 0](#_Toc412048014)

[Common errors 0](#_Toc412048015)

[**API Interface** 0](#_Toc412048016)

[Device Back-end Structure 0](#_Toc412048017)

[Settings.txt 0](#_Toc412048018)

[Session.txt 0](#_Toc412048019)

[Log.txt 0](#_Toc412048020)

[dataYYMM.jsn 0](#_Toc412048021)

[**Testing** 0](#_Toc412048022)

[**Ideas** 0](#_Toc412048023)

# **Introduction**

This system is running on Arduino Uno board.

## Why Arduino?

There are few other systems out there that I considered using such as Netduino and Raspberry Pi.  
  
Netduino has advantage of running Microsoft .NET framework, which is much easier to code in and debug. Problem is that not many sensors are compatible with the system since they require drivers for .NET libraries. That was the main reason I opted out this system. Minor note that Netduino costs as much as both Pi and Arduino together.  
  
Raspberry Pi can multi-task (while Arduino cannot), it is 40 times faster, has 128,000 times more RAM, and it is independent computer that can run on actual operating system, connect wirelessly and has multiple USP ports. In hardware department Pi is much more powerful than Arduino, but reality is that that power only reflect in software applications. Arduino’s simplicity makes it a much better bet for pure hardware projects.   
  
So I decided to go with Arduino board since it is has Real Time and Analog capability that other boards do not. This flexibility allows it to work with just about any kind of sensor and chips!!   
Arduino is much harder to break, for example you can unplug it and plug it whenever you want, while the Pi setup can be damaged by unplugging it without proper shutdown.

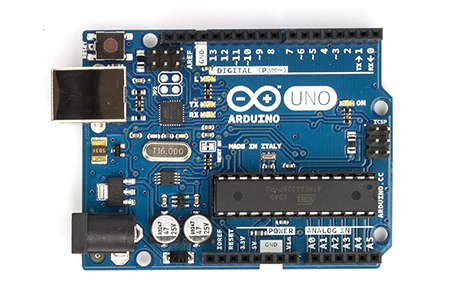
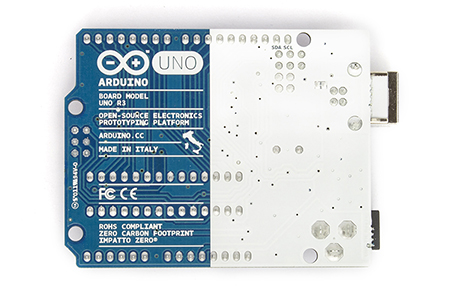
In short Pi is different kind of system that compliments Arduino (not compete with it), while Netduino provide abstraction through .NET which disconnect you from direct hardware communication. At this time this prevent it from using variety of sensors available on the market.

## Components

System is composed of:

* Arduino Uno R3 Board
* Add-on Shields (Ethernet Shield R3)
* Sensors
* Arduino code and libraries
* Website

### Arduino Uno R3 Board

|  |  |
| --- | --- |
| 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 |
| Length | 68.6 mm |
| Width | 53.4 mm |
| Weight | 25 g |

**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.

The ATmega328 has 32 KB FLASH (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).  
FLASH can be written to 10,000 times, and EEPROM 100,000 times before suffer from amnesia.

**The Input pins:**

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.

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.

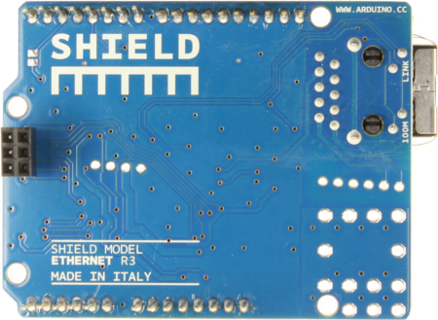
**Communication supported:**

**SPI:** The Arduino software includes a SPI library **I2C (TWI):** The Arduino software includes a Wire library to simplify use of the I2C bus**.  
Serial:** UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). 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). **1Wire:**

### Add-on Shields (Ethernet Shield R3)

I decided to use this one instead of WiFi one since at our house we do not broadcast our SID network. WiFi shield require our network name to broadcast and I was not sure if I wanted to change that. I could get another routher that broadcast its name for internal network but I did not think was worth it. So I decided to go with this one that is hardwired to the network.

Addon shield is consider as Arduino component that can stack on top of the Arduino board.  
System is using Arduino Ethernet Shield which also comes with an SD card

The Wiznet W5100 provides a network (IP) stack capable of both TCP and UDP, plus SD card.   
It supports up to **four** simultaneous socket connections. 

Arduino communicates with both the W5100 and SD card using the **SPI bus** (through the ICSP header) SPI bus uses 3 pins D11, D12 and D13.  
When working with SD library, SS is on Pin 4 (D4).  
When working with W5100 (Ethernet) library, SS is on Pin 10 (D10).  
All above pins cannot be used for general I/O.

Note that because the W5100 and SD card share the SPI bus, only one can be active at a time.  
If you are using both peripherals in your program, this should be taken care of by the corresponding libraries. If you're not using one of the peripherals in your program, however, you'll need to explicitly deselect it. To disable the SD card, set pin 4 as an output and write a high to it. For disabling the W5100, set digital pin 10 as a high output.

* PWR: indicates that the board and shield are powered
* LINK: indicates the presence of a network link and flashes when the shield transmits or receives data
* FULLD: indicates that the network connection is full duplex
* 100M: indicates the presence of a 100 Mb/s network connection (as opposed to 10 Mb/s)
* RX: flashes when the shield receives data
* TX: flashes when the shield sends data
* COLL: flashes when network collisions are detected

The solder jumper marked "INT" can be connected to allow the Arduino board to receive interrupt-driven notification of events from the W5100, but this is not supported by the Ethernet library. The jumper connects the INT pin of the W5100 to **digital pin 2** of the Arduino.

### Sensors

There are many kinds and types of sensors out there.   
Ideally I would like to implement no-contact sensors because they require minimum maintenance in the future.

**Input sensors:**  
- Air Temperature/Humidity  
- Real Time Clock (RTC) with bult in temperature sensor (good to measure temperature of the Arduino Device housing)  
- Water temperature(s) at different points in the pool and beds  
- Light sensor(s)  
- Water level sensor(s) for both pool and syphons  
- Motion sensors for water flow sensor(s)  
- Door open/closed sensor

**Output devices:**   
- LCD   
- light diodes as warning lights or status  
- Relays (to start/stop external devices as backup pumps, water heaters, warm lights, warning lights etc.)  
- Servo motor to open roof door  
- Call external services to send emails or texts using Ethernet  
- Log sensor to log files using SD card on Ethernet shield  
- Remotely start automatic fish feeders, or other devices through servo or regular motor.

**Input and Output expansions:**  
- shift registers as 74HC595N for additional 8 digital outputs using only 3 digital inputs using SPI bus.  
- MCP3008 for additional 8 inputs (10bit analog inputs) by using 3 digital pins and 1 SS (Slave Select pin) on existing SPI bus. (Therefore only one extra pin)

More in depth details about each sensor see Sensors section

### Arduino Code and Libraries

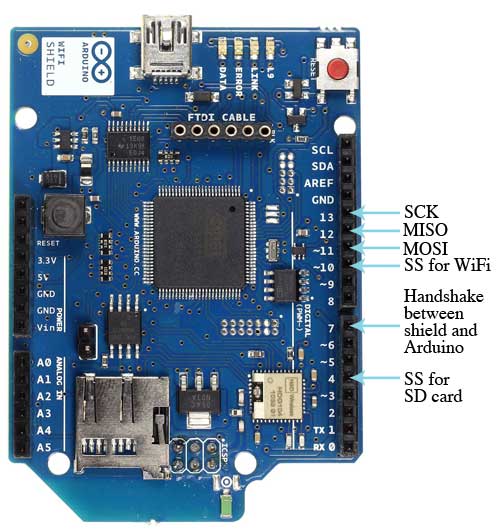
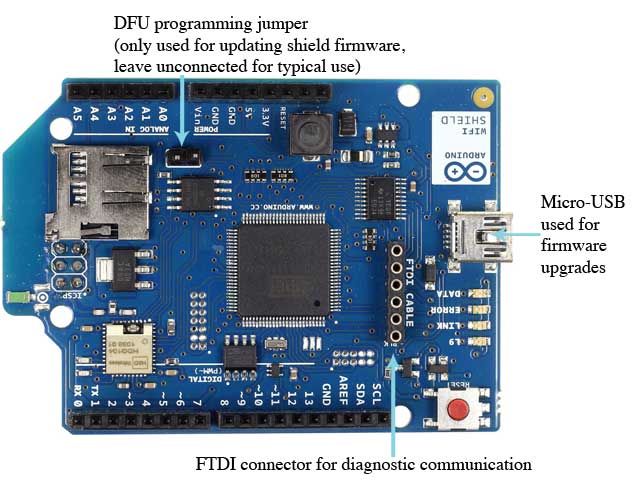
Most libraries are supplied by Arduino however there are some twists.  
- Ethernet shield with SD card has SD and Ethernet library, however wehen using SD interface they don’t get along, so hat to use lower level helper library: SdFat and SdFatUtil  
- Liquid Crystal library works for connecting directly to the Arduino board – but that uses too many digital pins. To use LCD through the SPI bus there is updated Liquid Crystal library found here:  
-

### Website

Decided to go with bootstrap instead of Jquery mobile. Mostly because of simplicity.  
I load css and js libraries externally through cdn servers since wanted to offload Arduino from serving those files from SD card.

# **Other Shields**

### WiFiShield



The shield can connect to encrypted networks that use either WPA2 Personal or WEP  
Encryption. It can also connect to open networks.

A network must broadcast its SSID for the shield to be able to connect.

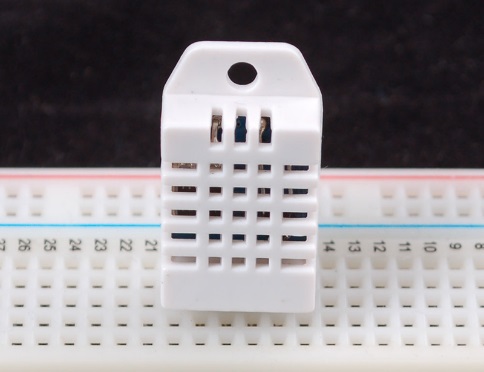
The Arduino WiFi Shield allows an Arduino board to connect to the internet using the 802.11 wireless specification (WiFi). It is based on the [HDG204](http://pub.ucpros.com/download/1451_hdg204_datasheet_pa4.pdf?osCsid=mcrh728ovgeg6ub4ka6mccrso5s) Wireless LAN 802.11b/g System in-Package. An AT32UC3 provides a network (IP) stack capable of both TCP and UDP.

# **Sensors**

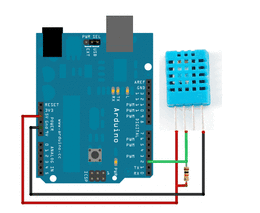
## Air Humidity and Temperature Sensor

**Detail Description:**Vktech Home Appliance DHT22/AM2302 Digital Temperature And Humidity Measurement Sensor  
Temperature Humidity Sensor (ASONG AM2302, 20366 B09D06, Jan 9 2015)

[**DHT22**](http://www.adafruit.com/products/385) **(DHT lib)**

* Low cost
* 3 to 5V power and I/O
* 2.5mA max current use during conversion (while requesting data)
* Good for 0-100% humidity readings with 2-5% accuracy
* Good for -40 to 125°C temperature readings ±0.5°C accuracy
* No more than 0.5 Hz sampling rate (once every 2 seconds)
* Body size 15.1mm x 25mm x 7.7mm
* 4 pins with 0.1" spacing

Likewise, it is fairly easy to connect up to the DHT sensors. They have four pins

* VCC (3 to 5V power)
* Data out
* Not connected
* Ground

Simply ignore pin 3, its not used. You will want to place a 10K resistor between VCC and the data pin, to act as a medium-strength pull up on the data line. The Arduino has built in pullups you can turn on but they're very weak, about 100K

This diagram shows how we will connect for the testing sketch. Connect data to pin 2, you can change it later to any pin.

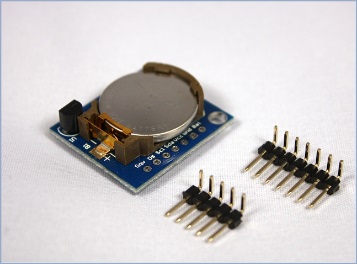
<https://github.com/adafruit/DHT-sensor-library>

**Summary**:  
Uses 1(any) Digital pin for input to measure both Temperature AND Humidity.

## Real Time Clock (RTC)

**Detail Description:**SainSmart Tiny RTC I2C DS1307 AT24C32 24C32 memory Real Time Clock Module for Arduino

**DS1307 (RTC lib)**

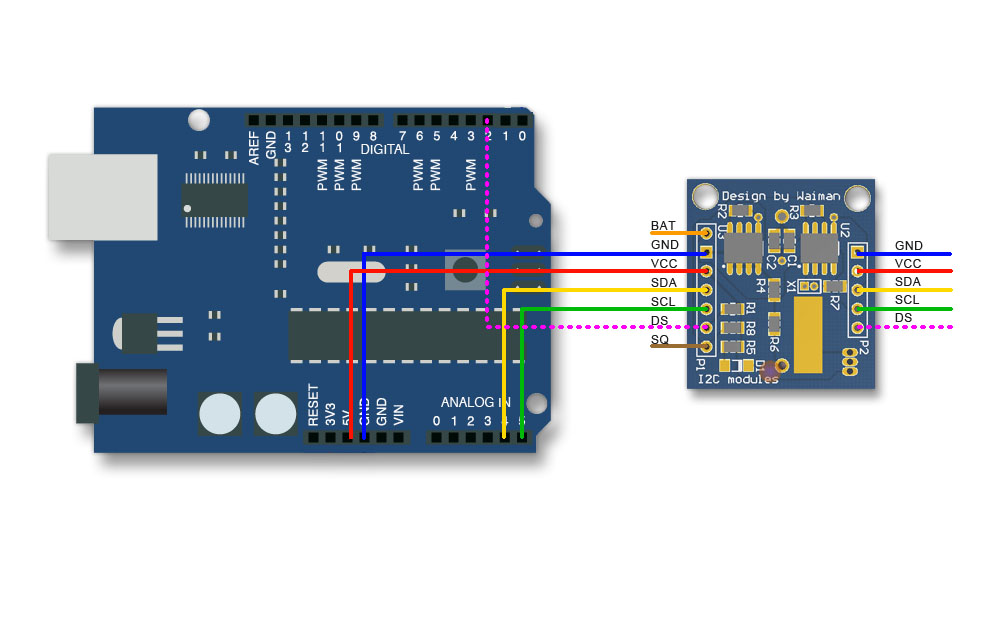
* **24C32** 32K I2C EEPROM memory.
* **DS1307** based RTC with CR2032 battery (Battery included).
* Fully charged, it can provide the DS1307 timing 1.
* Compact design, 27mm \* 28mm \* 8.4mm.
* Leads to the a DS1307 clock pin, to provide the clock signal for the microcontroller.

This is a great battery-backed real time clock (RTC) that allows your microcontroller project to keep track of time even if it is reprogrammed, or if the power is lost. Perfect for datalogging, clock-building, time stamping, timers and alarms, etc. The DS1307 is the most popular RTC, and works best with 5V-based chips such as the Arduino.

This breakout board is a kit and requires some light soldering which should only take about 15 minutes.

It can count leap-years and knows how many days are in a month, but it doesn't take care of Daylight Savings Time (because it changes from place to place)  
It auto switch to battery if there is no power.

You MUST have a coin cell installed for the RTC to work, if there is no coin cell, you should pull the battery pin low.

**You MUST have a coin cell installed for the RTC to work, if there is no coin cell, it will act strangly and possibly hang the Arduino so ALWAYS make SURE there's a battery installed, even if its a dead battery.**

Uses I2C interface:  
Uses 2 pins for the I2C bus: A4 (SDA) and A5 (SCL)  
Above pins can only be used for another I2C peripherals. (LCD screen for example)  
~~Needs 4.7 KOhm pull up resistor.~~ (Arduino built pull up resistors in the board)

Arduino Tiny RTC I2C Real Time Clock Pinout

|  |  |  |
| --- | --- | --- |
| PIN | Description | Comment |
| BAT | Battery voltage | To monitor the battery voltage, or not connected |
| GND | Ground | Ground |
| VCC | 5V supply | Power the module and charge the battery |
| SDA | I2C data | I2C data for the RTC |
| SCL | I2C clock | I2C clock for the RTC |
| DS | DS18B20 Temp. Sensor output | One wire interface |
| SQ | Square wave output | Normally not used |

|  |
| --- |
| batteryVoltageRead = analogRead (A0);  float batteryVoltage =float( batteryVoltageRead \* (5/1023.) );  // format might need some tweaking  Serial.println (batteryVoltage, 2);  // same here, not sure how to specify how many decimal points are shown |

The I2C wires "SDA" and "SCL" are the data line and clock line, they should be connected to the corresponding pins depending on the Arduino board.

|  |  |
| --- | --- |
| Board | I2C / TWI pins |
| Uno, Ethernet | A4 (SDA), A5 (SCL) |
| Mega2560 | 20 (SDA), 21 (SCL) |
| Leonardo | 2 (SDA), 3 (SCL) |
| Due | 20 (SDA), 21 (SCL), SDA1, SCL1 |

**Summary**:  
Uses 2 Analog pins for I2C bus. A4 and A5  
Uses 1 more digital pins if you want temperature through one wire interface  
Uses 1 wire for battery status.  
Need 4.7 KOhm pull up resistor

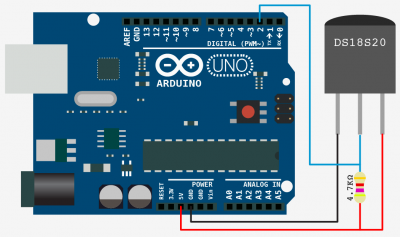
## Waterproof Digital Sensors

Vktech 5pcs 2M Waterproof Digital Temperature Temp Sensor Probe DS18b20

The DS18B20 is recommended for any application that requires 9 to 12 bits of temperature resolution. This device offers much more flexibility and is easier to use than the DS18S20

<http://milesburton.com/Main_Page?title=Dallas_Temperature_Control_Library>

<https://github.com/milesburton/Arduino-Temperature-Control-Library>

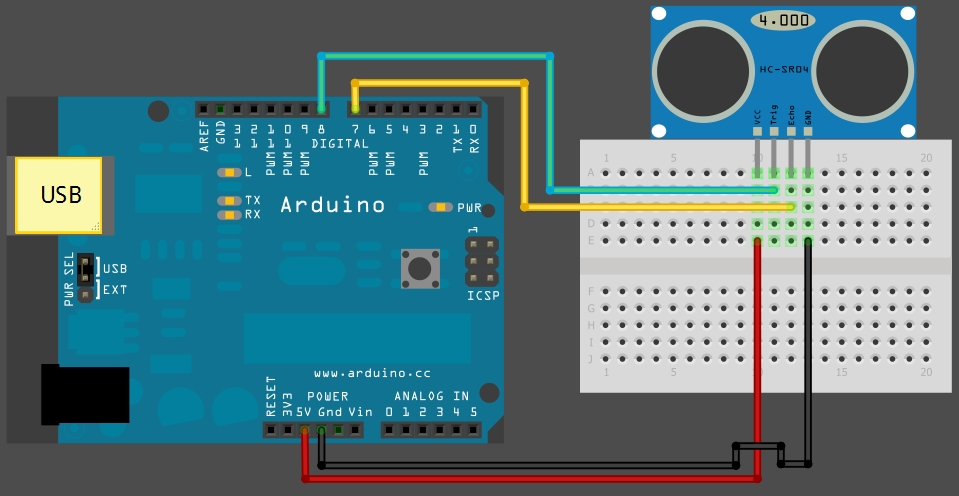


OneWire works ☺  
Dallas Temperature Control Library doesn’t (compile errors – remove const from function parameters fixes it!)  
Can use up to 255 sensors on same pin in parallel!!

Used sketch\_08\_03\_OneWire\_DS18B20\_2   
Save in settings if any sensor was not initialized! Then reboot remotely  
If sensor disconnected it shows value -127, and both 85?

## Ultra-Sonic Distance Sensor

SunFouder 2 pcs Ultrasonic Module HC-SR04 Distance Sensor for Arduino UNO MEGA R3 Mega2560 Due milanove Nano Robot XBee ZigBee



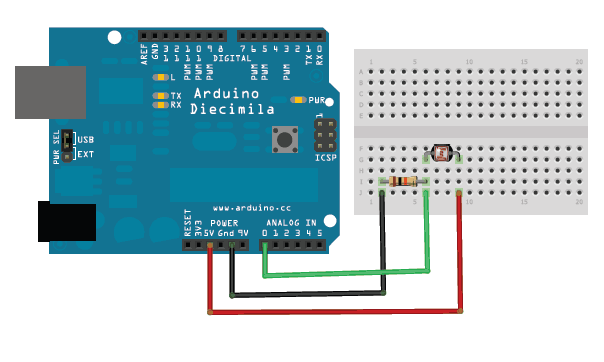
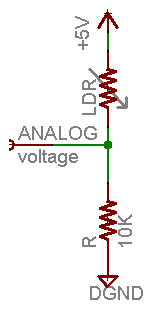
http://arduinobasics.blogspot.com/2012/11/arduinobasics-hc-sr04-ultrasonic-sensor.html

http://www.instructables.com/id/Simple-Arduino-and-HC-SR04-Example/

## Light Sensor

Photo Cell

https://learn.adafruit.com/photocells/using-a-photocell

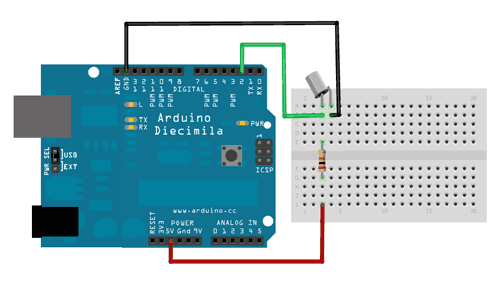


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ambient light like…** | **Ambient light (lux)** | **Photocell resistance (Ω)** | **LDR + R (Ω)** | **Current thru LDR +R** | **Voltage across R** |
| **Dim hallway** | **0.1 lux** | 600KΩ | 610 KΩ | 0.008 mA | 0.1 V |
| **Moonlit night** | **1 lux** | 70 KΩ | 80 KΩ | 0.07 mA | 0.6 V |
| **Dark room** | **10 lux** | 10 KΩ | 20 KΩ | 0.25 mA | 2.5 V |
| **Dark overcast day / Bright room** | **100 lux** | 1.5 KΩ | 11.5 KΩ | 0.43 mA | 4.3 V |
| **Overcast day** | **1000 lux** | 300 Ω | 10.03 KΩ | 0.5 mA | 5V |

## PH Sensor

https://www.sparkfun.com/products/10972   
https://github.com/OpenHydroponics/Billie-s-Hydroponic-Controller/blob/master/HydroponicControllerV1.1.0

## Tilt Switch

Tilt sensors are switches that can detect basic motion/orientation. The metal tube has a little metal ball that rolls around in it, when its tilted upright, the ball rolls onto the contacts sticking out of end and shorts them together.  
It is easy to Code since it is just a switch! (might need resistor also so you don’t short your input.

# **Outputs**

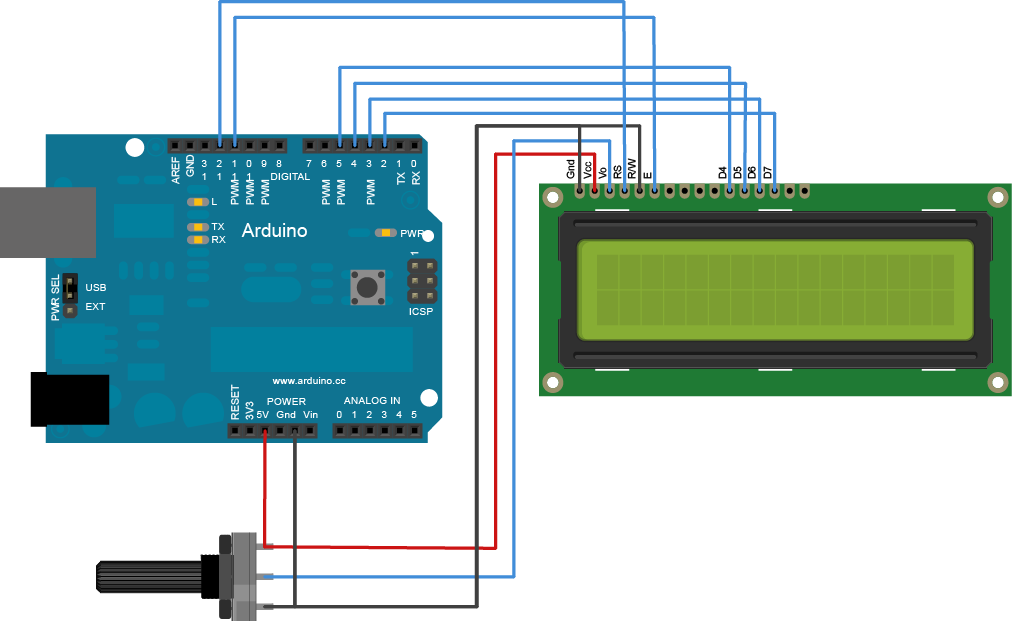
## LCD Screen

RioRand™ LCD Module for Arduino 20 x 4, White on Blue **HD44780**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | LCD Pin # | Function | | 1 | VSS (Gnd) | | 2 | VDD (+5V) | | 3 | Contrast Adjustment | | 4 | RS Register Select Input | | 5 | R/W Read/Write Signal,  normally at Gnd | | 6 | E Enable | | 7 | DB0 | | 8 | DB1 | | 9 | DB2 | | 10 | DB3 | | 11 | DB4 | | 12 | DB5 | | 13 | DB6 | | 14 | DB7 | | 15 | Back Light Adjustment | | 16 | LED (-) Gnd for back light | | |  |  | | --- | --- | | LCD Pin | Arduino Pin | | RS | D12 | | E | D11 | | DB4 | D5 | | DB5 | D4 | | DB6 | D3 | | DB7 | D2 | |

The Arduino connections are as follows:

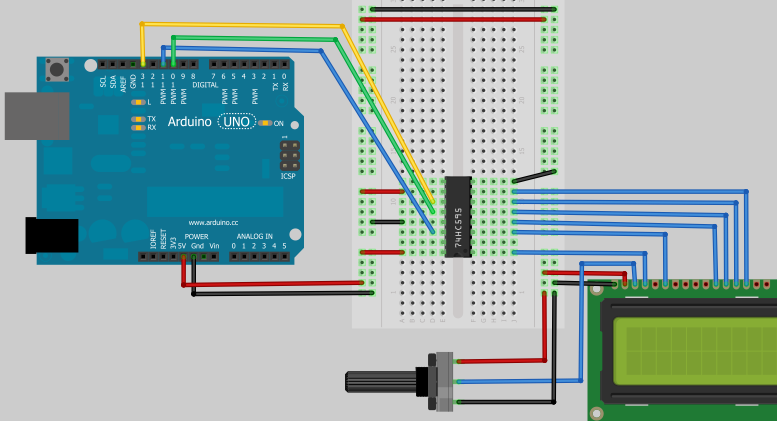
#include <LiquidCrystal.h> //Makes available the Arduino environment Liquid Crystal Display  
Liquid Crystal lcd(12,11,5,4,3,2); //Assigns the proper pin connections (as above) between Arduino and the display

Line 1 (top line): Addresses 0 -19  
Line 2: Addresses 64 - 83  
Line 3: Addresses 20 - 39  
Line 4: Addresses 84 - 95

You need 6 digitals pins to do stuff, so if you are planning to control several servo motors either get larger arduino like mega or buy I2C controller.

http://42bots.com/tutorials/arduino-controlled-lcd-using-a-shift-register-and-the-spi-library/   
<http://playground.arduino.cc/Main/LiquidCrystal>

used SS pin D8 instead.(D11 & D13)



# **Hardware Interfaces**

## SPI

## I2C

Uses 2 analog pins: A4 (SDA) and A5 (SCL)

## 1Wire

# **Hardware Design**

**Green House Arduino Uno**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | has to be on that pin | | | | | | | |
|  | can be any other pin | | | | | | | |
| DIGITAL | | D1 | D2 | D3 | D4 | D5 | D6 | D7 | | D8 | D9 | D10 | D11 | D12 | D13 |
| Network  & SD card (3 + 2 pins) | |  | INT? |  | SS2 |  |  |  | |  |  | SS1 | 1 | 2 | 3 |
| Temperature & Humidity | | in |  |  |  |  |  |  | |  |  |  |  |  |  |
| Clock | |  |  |  |  |  |  |  | |  |  |  |  |  |  |
| LCD (with SPI) | |  |  |  |  |  |  |  | | SS3 |  |  | 1 |  | 2 |
| Ultrasonic | |  |  |  |  | out | in |  | |  |  |  |  |  |  |
| Temperature (5) Wire-1 | |  |  | in |  |  |  |  | |  |  |  |  |  |  |
| Photo Cell | |  |  |  |  |  |  |  | |  |  |  |  |  |  |
| Relay | |  |  |  |  |  |  | out | |  |  |  |  |  |  |
| Servo Motor | |  |  |  |  |  |  |  | |  | out |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ANALOG | A1 | A2 | A3 | A4 | A5 |
| Network & SD card |  |  |  |  |  |
| Temperature & Humidity |  |  |  |  |  |
| Clock (I2C) |  |  |  | SDA | SCL |
| LCD |  |  |  |  |  |
| Ultrasonic |  |  |  |  |  |
| Temperature |  |  |  |  |  |
| Photo Cell |  |  | X |  |  |

|  |  |
| --- | --- |
| Input Sensors | Output |
| Water temperature bottom pool | Relay – start/stop water heater |
| Water temperature top pool |  |
| Air Humidity  Air Temperature | Servo motor – Open/Close roof (separate system?) |
| Pool Water Level | Send Warning (red lightbulb with relay?) |
| Photo Sensor | Warm Light On/Off (separate system?) |
| Clock | Servo Motor - Automatic Fish Feeder |
| SD Card | Logging / settings |
| Ethernet | Web Application – remote control |
| Camera (separate system) |  |
| Ph Sensor (too complicated – not available) | Send Warning |
| Monitor Clock Battery Voltage (enough pins?) | Send Warning |
| Flow Meter | Start backup pump? |
| Counter for sypher for each bed | Send Warning if not working |

# **Software Design**

## Web design

bootstrap instead of jquery mobile

## C design

Behind the scenes

Arduino Uno has:

**SRAM** (aka RAM). 2K

**PROGMEM** (aka FLASH) 32KB - 0.5KB (used by boot loader)

This is where your compiled program get stored.  
There are a finite number of write cycles (typically about **10,000**). Every time you upload program to Arduino, you did one write.

**EEPROM** 1KB

This is storage area that you can use.

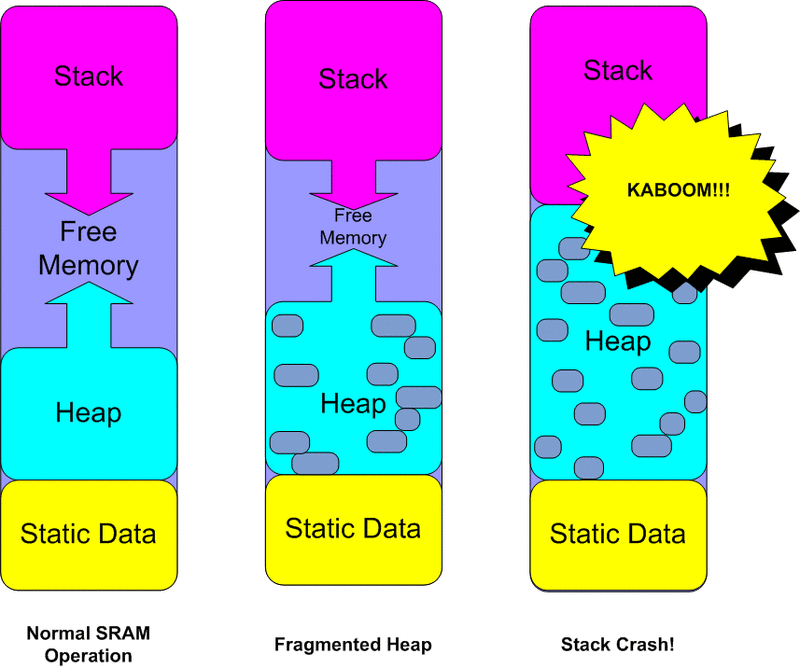
Note that while reads are unlimited, there are a finite number of write cycles (typically about **100,000**).To write to EEPR

Here is what happens to our program once is uploaded to Arduino:

1. When program is uploaded to Arduino, its compiled (binary) version is copied into **PROGMEM** (FLASH) memory. This is persistent storage
2. When program runs (when Arduino start up) it copies itself into the **SRAM**. This storage only lasts during power.
   * Global and Static variables get pushed to the start of the **heap** first (yellow on the pic below). They occupy SRAM forever (until reboot).
   * Dynamic allocations of objects fill **heap** second (blue).Dynamic allocations (malloc and free) can be de-allocated to free some space but this not necessarily cause the heap to shrink!  
     If dynamic data cannot fit in contiguous place in ram - it will reserve itself to the top of the heap! Left over holes cause heap to be **fragmented**
   * Local variables and parameters fill **stack** from the top. When function starts all parameters and local variables are stored to the stack and 100% released when function exists.

Notice that other libraries that you are using also use buffer and variables allocation, which takes up the SRAM as well in the same fashion.

SRAM allocation:



Ways to optimize

**Remove unused variables!**

If unsure: comment it out. If the sketch still compiles, get rid of it!

**Const vs #define vs PROGMEM**

|  |  |  |
| --- | --- | --- |
| #define | const | PROGMEM |
| #define SENSORPIN 8 digitalRead(SENSORPIN); | const int sensorPin = 8; digitalRead(sensorPin); | #include<avr/pgmspace.h> prog\_uchar sensorPin PROGMEM = 8; digitalRead(pgm\_read\_byte(&sensorPin)); |

In general const are preferred over #define.

#define is simply macro for find/replace option of pre-compiler, without looking into your code what are you trying to do.

With const you can define your variable type without compiler guessing it.

Either way in this case would get copied into **RAM**. Only way to use **no RAM** is using PROGMEM.

**F() Those Strings**

Serial.print("Hello World"); // This consumes RAM inside Static Data!!Terrible!!

Serial.print(F("Hello World")); // Keeps the character-array in PROGMEM aka FLASH

**More PROGMEM**

F() would not work inside some functions as strcpy() or strcat().

You would have to do little more work:

prog\_char string\_0[] PROGMEM = "Hello1";

prog\_char string\_1[] PROGMEM = "Hello2";

prog\_char string\_2[] PROGMEM = "Hello3";

prog\_char string\_3[] PROGMEM = "Hello4";

prog\_char string\_4[] PROGMEM = "Hello5";

// Then set up a table to refer to your strings.

PROGMEM const char \*string\_table[] = // change "string\_table" name to suit

{

string\_0,

string\_1,

string\_2,

string\_3,

string\_4,

string\_5 };

strcpy\_P(**buffer**, (char\*)pgm\_read\_word(&(string\_table[i]))); // Necessary casts and dereferencing, just copy.

Just make sure that **buffer** has necessary space allocated to it for copy to work.

Also floating point numbers in program memory do not appear to be supported.

prog\_char - a signed char (1 byte) -127 to 128

prog\_uchar - an unsigned char (1 byte) 0 to 255

prog\_int16\_t - a signed int (2 bytes) -32,767 to 32,768

prog\_uint16\_t - an unsigned int (2 bytes) 0 to 65,535

prog\_int32\_t - a signed long (4 bytes) -2,147,483,648 to \* 2,147,483,647.

prog\_uint32\_t - an unsigned long (4 bytes) 0 to 4,294,967,295

**EEPROM and EEPROMex**

**Remove Serial.print**

Serial takes lots of your ram. If not debugging remove it! Or u can use pin to turn on and off the debug messages ☺

if ( digitalRead ... ) Serial.print ...

This however might not help with you ram.

Better create function for logging and define #DEBUG variable that will cause compiler to skip the logging altogether so binary file does not even have serial logging in it.

**Libraries that use buffers**

Example: ....\Arduino-1.x.x\hardware\arduino\cores\arduino\HardwareSerial.cpp

Defines: #define SERIAL\_BUFFER\_SIZE 64

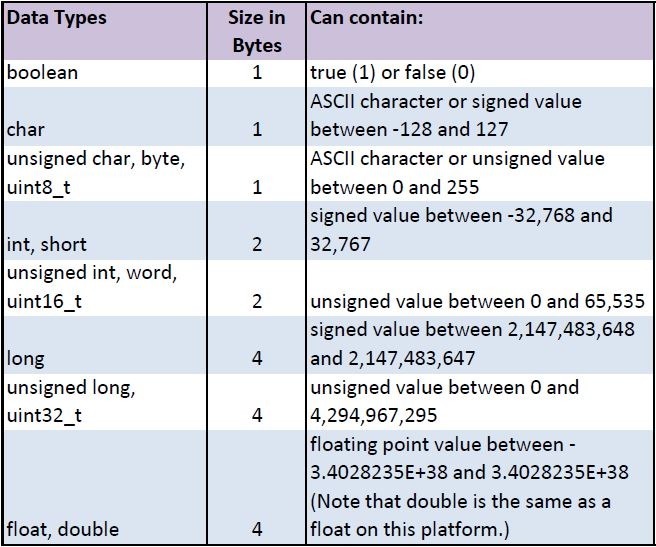
You could change this to 32 or less if your Arduino is not receiving lots of serial data.

**Dynamic allocations**

To avoid fragmented heap, malloc() large buffers only once at the start of the program in setup().

Do not use String variables since they do lots of memory allocations in the background.

**Reduce oversized variables**



Analog inputs: 0-1023 (0-5V) (A0, A1, A2, A3, A4, A5)  
Digital ~ outputs: 0-255 (0-5V) (~3, ~5, ~6, ~9, ~11)

Digital inputs (0-1) (0 for 0-2.5V and 1 for 2.5-5V)

**Common Misconceptions**

Variable name size

* + It doesn’t matte since compiler renames them

Comments in the code

* + Compiler removes all comments from the code when create binary version of the code

Numbers of functions

* + Breaking code into functions

### Common errors

**Scrambled Serial output**

If your serial output looks like this:

`3??f<ÌxÌ▯▯▯ü`³??f<

Make sure that your selected baud rate on the serial monitor matches the rate set by Serial.begin() in your sketch.

**Arduino hangs**

Most likely your heap and stack collided. Review above section for optimizing your code.

# **API Interface**

## Device Back-end Structure

Device uses following files on the SD card:

### Settings.txt

This is the smallest file and contains string of ascii bytes.

0-4 - Admin password (5 characters)  
5 - log internet traffic (‘1’=yes, ‘0’=no). Logged in **log.txt**.  
6 - log frequency that sensors should write (‘0’ = 10min, ‘1’ = 30min, ‘2’ = 60min) . Logged in **dataYYMM.jsn**.  
7 - log RAM usage (‘1’=yes, ‘0’=no) . Logged in **session.txt**.  
8 - reboot if SD not present (‘1’=yes, ‘0’=no)

Settings are loaded during device start up. If settings are not present, default values are used: “12345”, ‘1’ (yes),’2’ (60min), ’1’ (yes), ’0’ (no).  
Settings can also be changed on the fly using the API.

### Session.txt

Session file is used to write **start date/time** of the device during boot. Also device constantly write to this file current date/time so when device loses power this file contains start and **end date/time** of the current session.   
File also stores **sessionID** (not used at the moment). When session ends it is stored in the Log file, so that we can monitor how long device was running during each session.  
Also this file will contain **RAM usage** if settings are set to log that.  
If session file cannot be written to, device will reboot (or keep rebooting until can write to the file again) if the settings are set to reboot if SD not present.

### Log.txt

This file contains past **sessions** information (not the current one). It also logs **internet traffic** if it is enabled in the settings.

### dataYYMM.jsn

This file writes **sensor data**. YY is for current year, and MM is for current month.   
It is broken this way so that API can return current month of logs to optimize the speed.  
User would have to request previous months one at the time if would like to see older history.

# **Testing**

Length of cables that work with each type of sensor.

# **Ideas**

User set clock time  
User set frequency of sensor data logging  
User set range of temperatures, levels etc over api for alerts