***Traffic jam avoidance system***



**Prince Hussein bin Abdullah Faculty of Information Technology**

**Department of Computer Science**

Students:

Reema Mahmoud Hussein AL-Khawaja

Yazan Eizzaldeen Wahid Mutlaq

**Supervisor**

*Dr.: Wael Qassa*

*الاهداء*

*إلى من تجرعا الكأس فارغا ليسقياني قطرة حب*

*إلى من كلت أناملهما ليقدما لي لحظة سعادة*

*إلى من حصدا الأشواك عن دربي ليمهدا لي طريق العلم*

*إلى القلبين الكبيرين*

***أبي و أمي***

*وإلى رياحين حياتي اخوتي*

Contents

[Introduction 5](#_Toc470339396)

[Problem definition: 5](#_Toc470339397)

[Objectives: 6](#_Toc470339398)

[Abstraction and Motivation 6](#_Toc470339399)

[Methodology used: 6](#_Toc470339400)

[*1.* *Determine the hardware parts and programming language:* 7](#_Toc470339401)

[A. Arduino Uno : 7](#_Toc470339402)

[B. Reflective Optical Sensor with Transistor Output (TCRT5000) : 9](#_Toc470339403)

[*C.* *Grove - LCD RGB Backlight:* 11](#_Toc470339408)

[*D.* *Arduino Ethernet* 13](#_Toc470339412)

[*E.* *Programming language :* 15](#_Toc470339413)

[2. build each module and test seperatly: 16](#_Toc470339414)

[*A.* *Three Arduino UNO to implement master/slave:* 16](#_Toc470339415)

[*B.* *Three TCRT500 sensors in every street:* 26](#_Toc470339432)

[*C.* *Grove - LCD RGB Backlight :* 29](#_Toc470339438)

[*D.* *Arduino Ethernet:* 32](#_Toc470339442)

[*3.* *Integrate the whole system :* 43](#_Toc470339462)

[*4.* *Write code of whole system:* 44](#_Toc470339463)

[*5.* *Testing the project :* 50](#_Toc470339466)

[*REFERANCES :* 50](#_Toc470339467)

Introduction

In the Middle East and with technological revolution and evaluation this lead to increase number of people who depends on their own cars instead of using transportation.

All this reasons bring out a traffic jam; since we are part from developing countries it was ordered to open new streets or expand it very limited command, and takes long time to reach it.

And we as a student, we found the idea that limit that this each traffic jam and in many cases prevent them altogether. This project aims to provide software and hardware access for avoiding traffic jam, It’s also providing real time monitoring to provide the drivers current state of every fork and show the result to drivers before they going to choosing the fork to avoid choose the street that has traffic jam. This lead to limitation increasing traffic in the street that driver usual use it.

## Problem definition:

The problems of population explosion are becoming huge now and over that these traffic problems is really disturbing. The countries like KSA, Egypt where there are huge population problems will have to deal with these added problems now. It doesn't look easy to solve such problems unless nature itself maintains its balance. Having a car also becoming a problem for those who daily travel as it does take long time than regular time. During peak hours it is becoming very hard to move around the city and it would be better to take on public transport system instead of having this luxury of private vehicles. Some vehicles are also coming at cheaper rate making it reachable to every human being living in this world and so vehicles are everywhere now in these dense populated areas and it just looks like a jungle in peak hours and traveling through these huge traffic is becoming an adventure today.

# Objectives:

Based on the above problem we develop a system can sense the traffic on street rows and guide the driver to go through the less traffic row to reach his destination in shortest time .

## Abstraction and Motivation

The system built from special hardware equipment that facilitate manage with it, that programming in very simple and flexible way

The traffic jam avoidance system uses a special kind of receiver that receive a reflection of the led (that both within a same sensor) when some object is allocate over the sensor, the system detect it and provide the state of streets on special lcd to drivers so it can guide them to go through the less traffic street.

### Methodology used:

Out methodology consisit of:

1. **Deterimine project parts seperatly.**
2. **Build each module seperatly,andTest each module**
3. **Integrate the whole system**
4. **Write code of whole system.**
5. **Testing the whole project.**

# *Determine the hardware parts and programming language:*

## Arduino Uno :



ard a

The Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, 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.. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again. "Uno" means one in Italian and was chosen to mark the release of

Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards see the Arduino index of boards.

|  |  |
| --- | --- |
| Microcontroller | [ATmega328P](http://www.atmel.com/Images/doc8161.pdf) |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limit) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| PWM Digital I/O Pins | 6 |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 20 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32 KB (ATmega328P) of which 0.5 KB used by bootloader |
| SRAM | 2 KB (ATmega328P) |
| EEPROM | 1 KB (ATmega328P) |
| Clock Speed | 16 MHz |
| Length | 68.6 mm |
| Width | 53.4 mm |
| Weight |  |

# Reflective Optical Sensor with Transistor Output (TCRT5000) :



**Tcrt5000**

* ***TCRT5000 Datasheet:***

## DESCRIPTION:

The TCRT5000 is reflective sensors which include an infrared emitter and phototransistor in a leaded package which blocks visible light. The package includes two mounting clips. TCRT5000L is the long lead version.

### FEATURES:

• Package type: leaded

• Detector type: phototransistor

• Dimensions (L x W x H in mm): 10.2 x 5.8 x 7

• Peak operating distance: 2.5 mm • Operating range within > 20 % relative collector current: 0.2 mm to 15 mm

• Typical output current under test: IC = 1 mA

• Daylight blocking filter • Emitter wavelength: 950 nm

• Lead (Pb)-free soldering released

• Compliant to RoHS directive 2002/95/EC and in accordance to WEEE 2002/96/E

# APPLICATIONS:

• Position sensor for shaft encoder

• Detection of reflective material such as paper, IBM cards, magnetic tapes etc.

• Limit switch for mechanical motions in VCR

• General purpose - wherever the space is limited

## How It Works:

This sensor is basically composed of an LED emitting infrared (Light that in this frequency [Hz] is not visible to humans) and a phototransistor responsible for filtering natural light and capture infrared signals or not. It has a shield that separates the emitter and the receiver, depending on the reflectivity of the surface it is able to detect the color (On a BLACK and WHITE scale) that surface in question

# *Grove - LCD RGB Backlight:*



## Introduction :

This Grove enables you to set the color to whatever you like via the simple and concise Grove interface.

It takes I2C as communication method with your microcontroller.

So number of pins required for data exchange and backlight control shrinks from ~10 to 2, relieving IOs for

other challenging tasks. Besides, Grove - LCD RGB Backlight supports user-defined characters.

This product is a replacement of Grove - Serial LCD. If you are looking for primitive 16x2 LCD modules.

### Features

RGB Backlight

I2C communication

Built-in English fonts

16x2 LCD

# Specification:

Input Voltage: 5V

Operating Current : <60mA

CGROM : 10880 bit

CGRAM: 64\*8 bit

Colorful RGB Backlight

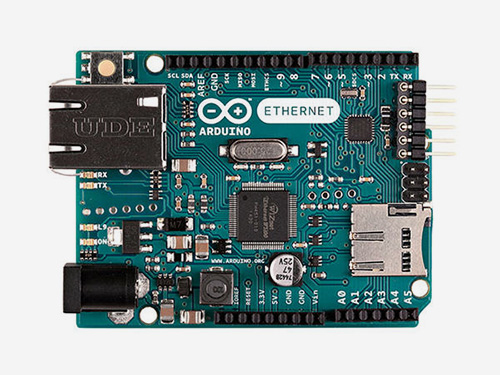
Built-in English and Japanese fonts

I2C communication uses only two IOs

Automatic power-on reset

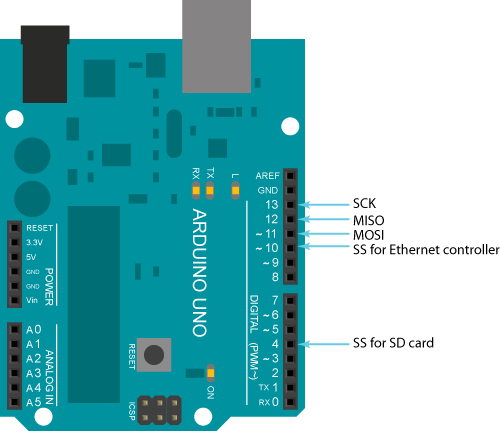
Suli-compatible Library

# *Arduino Ethernet*



With the Arduino Ethernet Shield, this library allows an Arduino board to connect to the internet. It can serve as either a server accepting incoming connections or a client making outgoing ones. The library supports up to four concurrent connection (incoming or outgoing or a combination).

Arduino communicates with the shield using the SPI bus. This is on digital pins 11, 12, and 13 on the Uno. pin 10 is used as SS.



The Arduino Ethernet Shield allows an Arduino board to connect to the internet. It is based on the Wiznet W5100 ethernet chip providing a network (IP) stack capable of both TCP and UDP. The Arduino Ethernet Shield supports up to four simultaneous socket connections. Use the Ethernet library to write sketches which connect to the internet using the shield.

The ethernet shield connects to an Arduino board using long wire-wrap headers which extend through the shield. This keeps the pin layout intact and allows another shield to be stacked on top..

The latest revision of the shield adds a micro-SD card slotwhich can be used to store files for serving over the network. It is compatible with the Arduino UNO (using the Ethernet library coming in Arduino 0019). An SD card library is not yet included in the standard Arduino distribution.

Arduino communicates with both the W5100 and SD card using the SPI bus (through the ICSP header). This is on digital pins 11,12and 13 on the Duemilanove and pins 5051and 52 on the Mega. On both boardspin 10 is used to select the W5100 and pin 4 for the SD card. These pins cannot be used for general i/o.

# *Programming language :*

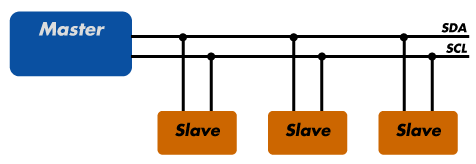
****

The main software in Arduino programing is Arduino Software IDE which is an open-source prototyping platform based on easy-to-use hardware and software. [Arduino boards](https://www.arduino.cc/en/Main/Products) are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the [Arduino programming language](https://www.arduino.cc/en/Reference/HomePage) (based on [Wiring](http://wiring.org.co/)), and [the Arduino Software (IDE)](https://www.arduino.cc/en/Main/Software), based on [Processing](https://processing.org/).

# build each module and test seperatly:

## *Three Arduino UNO to implement master/slave:*

* ***Master /slave :***

****

 Three boards are programmed to communicate with one another in a Master Reader/Slaves Sender configuration via the [I2C synchronous serial protocol](http://en.wikipedia.org/wiki/I2C). Several functions of Arduino's [Wire Library](https://www.arduino.cc/en/Reference/Wire) are used to accomplish this. Arduino 1, the Master, is programmed to request, and then read, a byte of data sent from the uniquely addressed Slave Arduino. Once that message is received, it can then be viewed in grove LCD.

The I2C( Inter-Integrated Circuit): protocol involves using two lines to send and receive data: a serial clock pin (SCL) that the Arduino Master board pulses at a regular interval, and a serial data pin (SDA) over which data is sent between the Slaves and Master. As the clock line changes from low to high (known as the rising edge of the clock pulse), a single bit of information - that will form in sequence the address of a specific device and a command or data - is transferred from the board to the I2C device over the SDA line. When this information is sent - bit after bit -, the called upon device executes the request and transmits its data back - if required - to the board over the same line using the clock signal still generated by the Master on SCL as timing. The initial eight bits (i.e. eight clock pulses) from the Master to Slaves contain the address of the device the Master wants data from. The bits after contain the memory address on the Slave that the Master wants to read data from or write data to, and the data to be written, if any

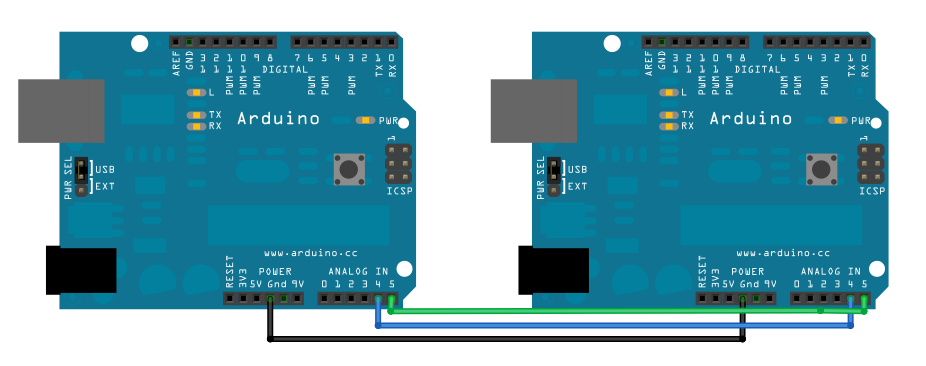
Each Slave device has to have its own unique address and both master and slave devices need to take turns communicating over the same data line. In this way, it's possible for your Arduino boards to communicate with many device or other boards using just two pins of your microcontroller, using each device's unique address.

# Hardware Required

* Arduino Boards
* hook-up wires

## Circuit

Connect pin 5 (the clock, or SCL, pin) and pin 4 (the data, or SDA, pin) on the master Arduino to their counterparts on the slave board. Make sure that both boards share a common ground. In order to enable serial communication, the slave Arduino must be connected to your computer via USB.



# Code:

## *Code for Master Reader:*

*// Demonstrates use of the Wire library*  
*// Reads data from an I2C/TWI slave device*

*// Refer to the "Wire Slave Sender" example for use with this*  
*// Created 29 March 2006*

*// This example code is in the public domain.*

#include <Wire.h>  
  
void **setup**() {  
  Wire.begin();        *// join i2c bus (address optional for master)*  
  Serial.begin(9600);  *// start serial for output*  
}  
  
void **loop**() {  
  Wire.requestFrom(8, 6);    *// request 6 bytes from slave device #8*  
  
  while (Wire.available()) { *// slave may send less than requested*  
    char c = Wire.read(); *// receive a byte as character*  
    Serial.print(c);         *// print the character*  
  }  
  
  delay(500);  
}

# ***Code for Slave Sender****:*

*// Demonstrates use of the Wire library*  
*// Sends data as an I2C/TWI slave device*  
*// Refer to the "Wire Master Reader" example for use with this*  
  
*// Created 29 March 2006*  
*// This example code is in the public domain.*  
  
#include <Wire.h>  
void **setup**() {  
  Wire.begin(8);                *// join i2c bus with address #8*  
  Wire.onRequest(requestEvent); *// register event*  
}  
  
void **loop**() {  
  delay(100);  
}  
*// function that executes whenever data is requested by master*  
*// this function is registered as an event, see setup()*  
void requestEvent() {  
  Wire.write("hello "); *// respond with message of 6 bytes* *// as expected by master*  
}

## Functions specification for master /slave:

## *Serial:*

Description:

Serial is used for communication between the Arduino board and a computer or other devices. You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().

# *begin()*

Description:

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200

Syntax: Serial.begin(speed)

Parameters: speed: in bits per second (baud) – long

## *print() , println()*

Description:

Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is. For example:

Serial.print(78) gives "78"

Serial.print(1.23456) gives "1.23"

Serial.print('N') gives "N"

Serial.print("Hello world.") gives "Hello world."

An optional second parameter specifies the base (format) to use; permitted values are BIN (binary, or base 2), OCT (octal, or base 8), DEC (decimal, or base 10), HEX (hexadecimal, or base 16). For floating point numbers, this parameter specifies the number of decimal places to use. For example:

Serial.print(78, BIN) gives "1001110"

Serial.print(78, OCT) gives "116"

Serial.print(78, DEC) gives "78"

Serial.print(78, HEX) gives "4E"

Serial.println(1.23456, 0) gives "1"

Serial.println(1.23456, 2) gives "1.23"

Serial.println(1.23456, 4) gives "1.2346"

You can pass flash-memory based strings to Serial.print() by wrapping them with F(). For example :

Serial.print(F(“Hello World”))

To send a single byte, use [Serial.write](https://www.arduino.cc/en/Serial/Write)().

Syntax:

Serial.print(val)   
Serial.print(val, format)

*Parameters*

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

*Returns:*

size\_t (long): print() returns the number of bytes written, though reading that number is optional

# *Wire Library:*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| This library allows you to communicate with I2C / TWI devices. On the Arduino boards with the R3 layout (1.0 pinout), the SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin. The Arduino Due has two I2C / TWI interfaces SDA1 andSCL1 are near to the AREF pin and the additional one is on pins 20 and 21.  As a reference the table below shows where I2C pins are located on various Arduino boards.   |  |  | | --- | --- | | **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 | |

# *Wire.begin(): , Wire.begin(address):*

*Description****:***

Initiate the Wire library and join the I2C bus as a master or slave. This should normally be called only once.

Parameters: address: the 7-bit slave address (optional); if not specified, join the bus as a master.

## *Wire.requestFrom()*

Description

Used by the master to request bytes from a slave device. The bytes may then be retrieved with the [*available()*](https://www.arduino.cc/en/Reference/WireAvailable)and [*read()*](https://www.arduino.cc/en/Reference/WireRead) functions.

As of Arduino 1.0.1, *requestFrom()* accepts a boolean argument changing its behavior for compatibility with certain I2Cdevices.

If true, *requestFrom()* sends a stop message after the request, releasing the I2C bus.

If false, *requestFrom()* sends a restart message after the request. The bus will not be released, which prevents another master device from requesting between messages. This allows one master device to send multiple requests while in control.

The default value is true.

*Syntax*

Wire.requestFrom(address, quantity)  
Wire.requestFrom(address, quantity, stop)

*Parameters*

address: the 7-bit address of the device to request bytes from

quantity: the number of bytes to request

stop : boolean. true will send a stop message after the request, releasing the bus. false will continually send a restart after the request, keeping the connection active.

*Returns*

byte : the number of bytes returned from the slave device

# *Wire.onRequest(handler):*

*Description*

Register a function to be called when a master requests data from this slave device.

*Parameters*

handler: the function to be called, takes no parameters and returns nothing, e.g.: void myHandler()

*Returns:*

None

# *Wire.available()*

*Description*

Returns the number of bytes available for retrieval with [read()](https://www.arduino.cc/en/Reference/WireRead). This should be called on a master device after a call to [requestFrom()](https://www.arduino.cc/en/Reference/WireRequestFrom) or on a slave inside the [onReceive()](https://www.arduino.cc/en/Reference/WireOnReceive) handler.

available() inherits from the [Stream](https://www.arduino.cc/en/Reference/Stream) utility class.

*Parameters*

None

*Returns*

The number of bytes available for reading.

## *Wire.read()*

*Description*

Reads a byte that was transmitted from a slave device to a master after a call to [requestFrom()](https://www.arduino.cc/en/Reference/WireRequestFrom) or was transmitted from a master to a slave. read() inherits from the [Stream](https://www.arduino.cc/en/Reference/Stream) utility class.

*Syntax*

Wire.read()

*Parameters*

none

*Returns*

The next byte received

# *Wire.write()*

*Description*

Writes data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device (in-between calls to beginTransmission() and endTransmission()).

*Syntax*

Wire.write(value)   
Wire.write(string)   
Wire.write(data, length)

*Parameters*

value: a value to send as a single byte

string: a string to send as a series of bytes

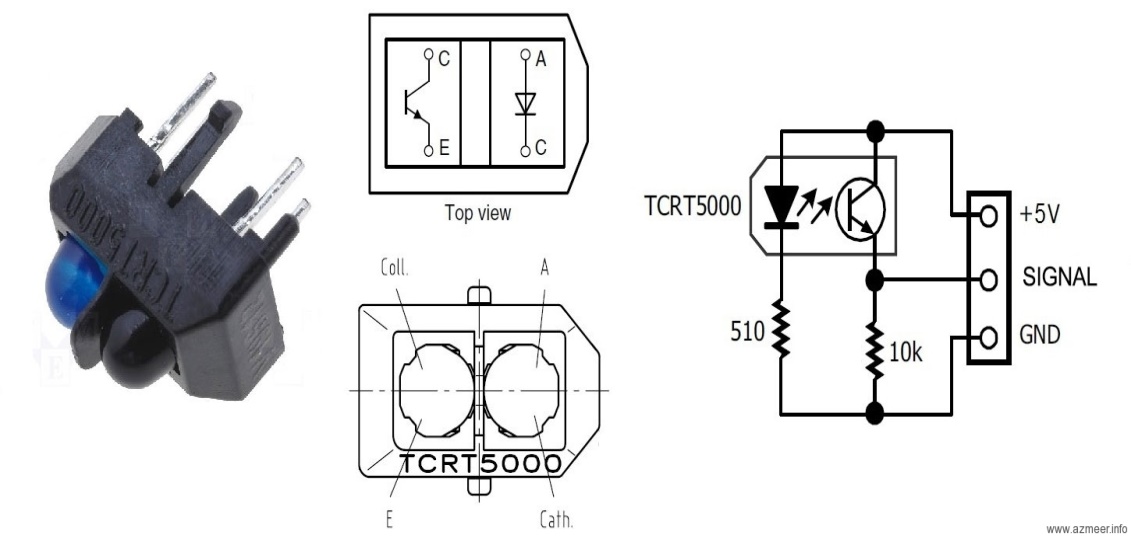
data: an array of data to send as bytes

length: the number of bytes to transmit

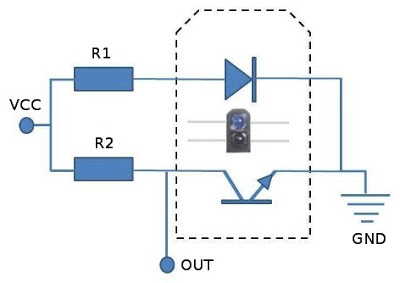
*Returns***:**

byte: write() will return the number of bytes written, though reading that number is optional

# *Three TCRT500 sensors in every street:*

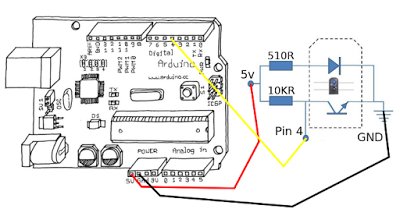


* Electrical Circuit:  
    
  This circuit is just one example of assembly and use of this sensor, you can change your settings to suit certain situations or specifications, always following what was specified in the DATASHEET.

[](http://3.bp.blogspot.com/-vHhAM9-XAG0/VcPhgAoyArI/AAAAAAAABwc/O9lvBeIUWHs/s1600/tcrt5000-schema.jpg)

Schematic Circuit

* Material:  
  1x - 10KR resistor (Brown, Black, Orange)  
  1x – 200R resistor (Green, Brown, Brown)
* Assembly  
    
  How LEDs and transistors are polarized, note the beveled side of the sensor to the correct position, incorrect junctions may damage the sensor.

[](http://4.bp.blogspot.com/-gdQQ3V8610I/VcPhw3Blu_I/AAAAAAAABwg/XZwndam14Qo/s1600/tcrt5000-arduino.png)

In the above circuit the resistor (10KR) is used as a "pull up" resistor. It ensures that the output is at a high level while the phototransistor is not conducting, it is what happens when there is no incidence of infrared light on it and prevents a short circuit when the phototransistor conducts.  
  
Increasing the value of (10KR) means, respecting the manufacturer limits , increase the sensitivity of the sensor, as it will be easier to put the phototransistor output at low level .Conversely, reducing the value of (10KR) it also reduces the sensitivity of the sensor.(Ideally placing a Trimpot "For example 20kR" in series with a resistor "For example 10KR" allows for an adjustment at runtime environment to this sensor)

## The bass code of TCRT5000 :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | int ledPin = 13; // Arduino' inner led  int sigPin = 4; // For sensor signal  int value = 0; // Holds the returned value    void setup(){    Serial.begin(9600);    pinMode(ledPin, OUTPUT); // Arduino's led as output    pinMode(sigPin, INPUT); // signal pin as input  }  void loop(){    value = digitalRead(sigPin); // Read the sensor      if (value == HIGH) { // iF HIGH, black area     digitalWrite(ledPin, HIGH); // Turn on the LED      Serial.write("Preto"); // Write on console    }  else { // Else, white area      digitalWrite(ledPin, LOW); // Turn off the LED      Serial.write("Branco"); // Write on console    }  } Functions specification: *pinMode():*  Description: Configures the specified pin to behave either as an input or an output  Syntax: pinMode(pin, mode)  Parameters:  pin: the number of the pin whose mode you wish to set  mode: [INPUT](https://www.arduino.cc/en/Reference/Constants) or [OUTPUT](https://www.arduino.cc/en/Reference/Constants)  *digitalWrite():*  Description: Write a[**HIGH**](https://www.arduino.cc/en/Reference/Constants)or a[**LOW**](https://www.arduino.cc/en/Reference/Constants)value to a digital pin  Syntax: digitalWrite(pin, value)  Parameters: pin: the pin number  Value: HIGH or LOW *Grove - LCD RGB Backlight :*Demonstration : This demonstration will show you how to use Grove - LCD RGB Backlight, we need a Arduino else.   * Hardware Installation   Hardware installation is very easy, because there's an I2C Grove in Arduino, so what we need to do is Connect Grove cable with pin 5 (the clock , or SCL pin) , pin 4 (the data , or SDA pin) and as usual connect the GND and VCC  نتيجة بحث الصور عن ‪connect grove lcd rgb backlight with arduino‬‏   * Download grove library:   You can download the grove library and then it work directly with the Arduino software, shields and IDEs Code:   #include <Wire.h>  #include "rgb\_lcd.h"  rgb\_lcd lcd;  const int colorR = 255;  const int colorG = 0;  const int colorB = 0;  void setup()  {  // set up the LCD's number of columns and rows:  lcd.begin(16, 2);    lcd.setRGB(colorR, colorG, colorB);    // Print a message to the LCD.  lcd.print("hello, world!");  delay(1000);  }  void loop()  {  // set the cursor to column 0, line 1  // (note: line 1 is the second row, since counting begins with 0):  lcd.setCursor(0, 1);  // print the number of seconds since reset:  lcd.print(millis()/1000);  delay(100);  } Functions specification: ***( #include "rgb\_lcd.h")*** : import grove library  ***( lcd.begin(16, 2) ) :*** Initiate the grove library. This should normally be called only once.  Parameters: dimension of monitor.  ***(lcd.setRGB(R,G,B)):*** set a background to grove lcd.  ***( lcd.setCursor(C, R) ;):*** set the cursor to print .  Parameter : first parameter to determine the column to start print , and the second parameter to determine row to start print .  ***( lcd.print(“hello world”) ) :*** Print a message to the LCD.  Parameter: sentence that would be print . *Arduino Ethernet:*Getting Started Use the [Arduino Software (IDE)](https://www.arduino.cc/en/Main/Software), and start to tinker with coding and electronics.  https://www.arduino.cc/en/uploads/Guide/ArduinoWithEthernetShield.jpg Connecting the Shield To use the shield, mount it on top of an Arduino board (e.g. the Uno). To upload sketches to the board, connect it to your computer with a USB cable as you normally would. Once the sketch has been uploaded, you can disconnect the board from your computer and power it with an external power supply.  The Ethernet shield allows you to connect a WizNet Ethernet controller to the Arduino or Genuino boards via the SPI bus. It uses pins 10, 11, 12, and 13 for the SPI connection to the WizNet. Later models of the Ethernet shield also have an SD Card on board. Digital pin 4 is used to control the slave select pin on the SD card.  The shield should be connected to a network with an ethernet cable. You will need to change the network settings in the program to correspond to your network.  نتيجة بحث الصور عن ‪ethernet with arduino connect‬‏ Network Settings The shield must be assigned a MAC address and a fixed IP address using the [Ethernet.begin()](https://www.arduino.cc/en/Reference/EthernetBegin) function. A MAC address is a globally unique identifier for a particular device. Current Ethernet shields come with a sticker indicating the MAC address you should use with them. For older shields without a dedicated MAC address, inventing a random one should work, but don't use the same one for multiple boards. Valid IP addresses depend on the configuration of your network. It is possible to use DHCP to dynamically assign an IP to the shield. Optionally, you can also specify a network gateway and subnet. A Brief Introduction to the Serial Peripheral Interface (SPI) Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.  With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices:   * MISO (Master In Slave Out) - The Slave line for sending data to the master, * MOSI (Master Out Slave In) - The Master line for sending data to the peripherals, * SCK (Serial Clock) - The clock pulses which synchronize data transmission generated by the master   and one line specific for every device:   * SS (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.  Code */\*   Web Server   A simple web server that shows the value of the analog input pins.  using an Arduino Wiznet Ethernet shield.   Circuit:  \* Ethernet shield attached to pins 10, 11, 12, 13  \* Analog inputs attached to pins A0 through A5 (optional)*   #include <SPI.h> #include <Ethernet.h>  *// Enter a MAC address and IP address for your controller below.* *// The IP address will be dependent on your local network:* byte mac[] = {   0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; IPAddress ip(192, 168, 1, 177);  *// Initialize the Ethernet server library* *// with the IP address and port you want to use* *// (port 80 is default for HTTP):* EthernetServer server(80);  void **setup**() {   *// Open serial communications and wait for port to open:*   Serial.begin(9600);   while (!Serial) {     ; *// wait for serial port to connect. Needed for native USB port only*   }     *// start the Ethernet connection and the server:*   Ethernet.begin(mac, ip);   server.begin();   Serial.print("server is at ");   Serial.println(Ethernet.localIP()); }   void **loop**() {   *// listen for incoming clients*   EthernetClient client = server.available();   if (client) {     Serial.println("new client");     *// an http request ends with a blank line*     boolean currentLineIsBlank = true;     while (client.connected()) {       if (client.available()) {         char c = client.read();         Serial.write(c);         *// if you've gotten to the end of the line (received a newline*         *// character) and the line is blank, the http request has ended,*         *// so you can send a reply*         if (c == '**\n**' && currentLineIsBlank) {           *// send a standard http response header*           client.println("HTTP/1.1 200 OK");           client.println("Content-Type: text/html");           client.println("Connection: close");  *// the connection will be closed after completion of the response*           client.println("Refresh: 5");  *// refresh the page automatically every 5 sec*           client.println();           client.println("<!DOCTYPE HTML>");           client.println("<html>");           *// output the value of each analog input pin*           for (int analogChannel = 0; analogChannel < 6; analogChannel++) {             int sensorReading = analogRead(analogChannel);             client.print("analog input ");             client.print(analogChannel);             client.print(" is ");             client.print(sensorReading);             client.println("<br />");           }           client.println("</html>");           break;         }         if (c == '**\n**') {           *// you're starting a new line*           currentLineIsBlank = true;         } else if (c != '**\r**') {           *// you've gotten a character on the current line*           currentLineIsBlank = false;         }       }     }     *// give the web browser time to receive the data*     delay(1);     *// close the connection:*     client.stop();     Serial.println("client disconnected");   } } Function specification |

***IPAddress()***

Description

Defines an IP address. It can be used to declare both local and remote addresses.

Syntax

IPAddress(address);

Parameters

address: a comma delimited list representing the address (4 bytes, ex. 192, 168, 1, 1)

Returns

None

***EthernetServer()***

Description

Create a server that listens for incoming connections on the specified port.

Syntax

Server(port);

Parameters

port: the port to listen on (int)

Returns

None

***Ethernet.begin()***

Description

Initializes the ethernet library and network settings.

With version 1.0, the library supports DHCP. Using Ethernet.begin(mac) with the proper network setup, the Ethernet shield will automatically obtain an IP address. This increases the sketch size significantly. To make sure the DHCP lease is properly renewed when needed, be sure to call [Ethernet.maintain()](https://www.arduino.cc/en/Reference/EthernetMaintain) regularly.

Syntax

Ethernet.begin(mac);   
Ethernet.begin(mac, ip);   
Ethernet.begin(mac, ip, dns);   
Ethernet.begin(mac, ip, dns, gateway);   
Ethernet.begin(mac, ip, dns, gateway, subnet);

Parameters

mac: the MAC (Media access control) address for the device (array of 6 bytes). this is the Ethernet hardware address of your shield. Newer Arduino Ethernet Shields include a sticker with the device's MAC address. For older shields, choose your own.

ip: the IP address of the device (array of 4 bytes)

dns: the IP address of the DNS server (array of 4 bytes). optional: defaults to the device IP address with the last octet set to 1

gateway: the IP address of the network gateway (array of 4 bytes). optional: defaults to the device IP address with the last octet set to 1

subnet: the subnet mask of the network (array of 4 bytes). optional: defaults to 255.255.255.0

Returns

The DHCP version of this function, Ethernet.begin(mac), returns an int: 1 on a successful DHCP connection, 0 on failure. The other versions don't return anything.

***begin()***

Description

Tells the server to begin listening for incoming connections.

Syntax

*server*.begin()

Parameters

None

Returns

None

***Ethernet.localIP()***

Description

Obtains the IP address of the Ethernet shield. Useful when the address is auto assigned through DHCP.

Syntax

Ethernet.localIP();

Parameters

none

Returns

the IP address

***EthernetClient()***

Description

Creates a client which can connect to a specified internet IP address and port (defined in the [client.connect()](https://www.arduino.cc/en/Reference/ClientConnect) function).

Syntax

EthernetClient()

Parameters

None

***available()***

Description

Gets a client that is connected to the server and has data available for reading. The connection persists when the returned client object goes out of scope; you can close it by calling *client*.stop().

Syntax

*server*.available()

Parameters

None

Returns

a Client object; if no Client has data available for reading, this object will evaluate to false in an if-statement (see the example below)

***connected()***

Description

Whether or not the client is connected. Note that a client is considered connected if the connection has been closed but there is still unread data.

Syntax

*client*.connected()

Parameters

none

Returns

Returns true if the client is connected, false if not.

***available()***

Description

Returns the number of bytes available for reading (that is, the amount of data that has been written to the client by the server it is connected to).

available() inherits from the [Stream](https://www.arduino.cc/en/Reference/Stream) utility class.

Syntax

*client*.available()

Parameters

none

Returns

The number of bytes available.

***read()***

Read the next byte received from the server the client is connected to (after the last call to read()).

read() inherits from the [Stream](https://www.arduino.cc/en/Reference/Stream) utility class.

Syntax

*client*.read()

Parameters

none

Returns

The next byte (or character), or -1 if none is available.

***println()***

Description

Print data, followed by a carriage return and newline, to the server a client is connected to. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters '1', '2', '3').

Syntax

*client*.println()   
*client*.println(data)   
*client*.print(data, BASE)

Parameters

data (optional): the data to print (char, byte, int, long, or string)

BASE (optional): the base in which to print numbers: DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Returns

byte: return the number of bytes written, though reading that number is optional

***stop()***

Description

Disconnect from the server.

Syntax

*client*.stop()

Parameters

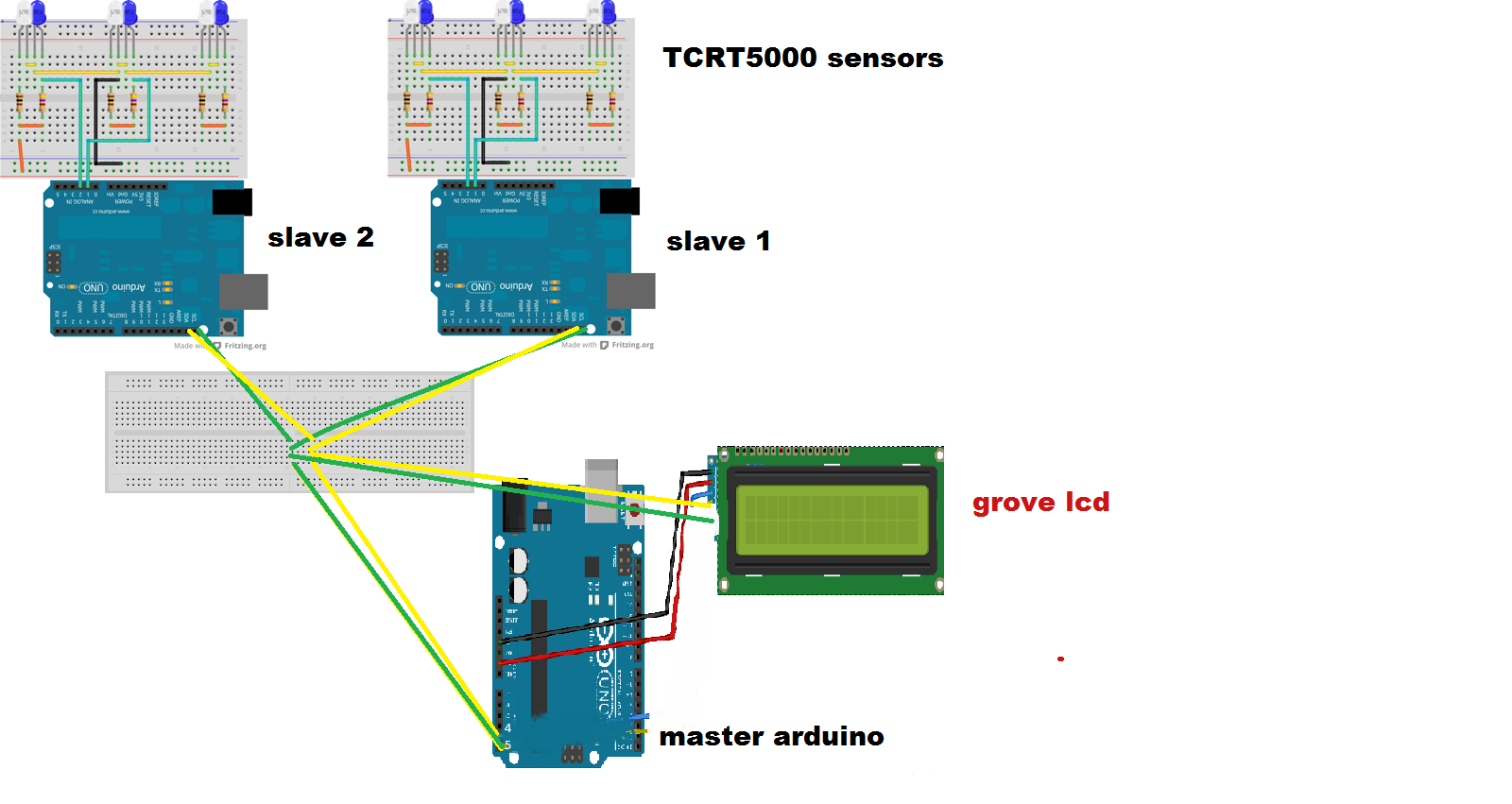
none

Returns : None

# *Integrate the whole system :*

At the end we integrate the project part which previously tested speratly .

The system does read every fork state in any zone via the TCRT5000 sensors which placed on the street , the arduino slaves send these states (via I2C protocol )to master arduino which grove lcd connects with , the information that received from slaves showing in grove lcd.



# *Write code of whole system:*

In this section we built the sketch on arduino software vresion 1.6.1 depinding on the circuit above, our code wrote as below

## Slave code

**// Wire Slave sender**

**//send data to Master via i2c**

#include <Wire.h>

int inPin1 = 2;

int inPin2 = 4;

int inPin3 = 7;

void setup() {

Serial.begin(9600); **// start serial for output**

pinMode(inPin1,INPUT); **//we define the pins as input**

pinMode(inPin2,INPUT);

pinMode(inPin3,INPUT);

Wire.begin(8); **// join i2c bus with address #8**

Wire.onRequest(requestEvent); **// register event**

}

void loop() {

Serial.println(digitalRead(inPin1)); **//print on serial monitor just for testing**

Serial.print(digitalRead(inPin2));

Serial.print(digitalRead(inPin3));

delay(500);

}

**// function that executes whenever data is requested by master  
// this function is registered as an event, see setup()**

void requestEvent() {

int read1=digitalRead(inPin1);  **//Reads the value from a specified digital pin**

int read2=digitalRead(inPin2);

int read3=digitalRead(inPin3);

int count=read1+read2+read3;

Wire.write(count);

}

# Master code

/\* Wire Master Reader

Demonstrates use of the Wire library

Reads data from an I2C/TWI slave device

Circuit:

\* Ethernet shield attached to pins 10, 11, 12, 13

\* Analog inputs attached to pins A0 through A5 (optional)

\*/

#include <SPI.h>

#include <Ethernet.h>

#include <Wire.h>

#include "rgb\_lcd.h"

// Enter a MAC address and IP address for your controller below.

// The IP address will be dependent on your local network:

byte mac[] = {

0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED

};

IPAddress ip(10, 107, 100, 250 );

// Initialize the Ethernet server library

// with the IP address and port you want to use

// (port 80 is default for HTTP):

EthernetServer server(80);

rgb\_lcd lcd;

const int colorR = 255; //for color

const int colorG = 0; //for color

const int colorB = 0; //for color

void setup() {

lcd.begin(16, 2); //Start grove-LCD to diplay

Wire.begin(); //join i2c bus (address optional for master)

Serial.begin(9600); // start serial for output

while (!Serial) {

; // wait for serial port to connect. Needed for native USB port only

}

// start the Ethernet connection and the server:

Ethernet.begin(mac, ip);

server.begin();

Serial.print("server is at ");

Serial.println(Ethernet.localIP());

lcd.setRGB(colorR, colorG, colorB); //set the bg\_color for LCD

}

void loop()

{

String A;

String B;

Wire.requestFrom(8,1); // request 1 bytes from slave device #8

lcd.setCursor(0, 0); //set cursor for grove-LCD

while (Wire.available()) //if slave connected or available

{

int val=Wire.read(); //read from slave device #8

if (val == 0 || val == 1)

{

String x="Street1 is Full!";

//lcd.write(x); //write on display

A=x;

}

else

{

String x1="Street1 is Empty";

A=x1;

}

char a[20];

A.toCharArray(a,20);

lcd.write(a);

}

Wire.requestFrom(9,1); // request 1 bytes from slave device #9

lcd.setCursor(0,1); //set cursor for grove-LCD

while (Wire.available()) //if slave connected or available

{

int val=Wire.read();

if (val == 0 || val == 1)

{

String y ="Street2 is Full!";

B=y;

}

else

{

String y1="Street2 is Empty";

B=y1;

}

char b[20];

B.toCharArray(b,20);

lcd.write(b);

}

EthernetClient client = server.available();

if (client) {

Serial.println("new client");

// an http request ends with a blank line

boolean currentLineIsBlank = true;

while(client.connected()){

if (client.available()) {

char c = client.read();

Serial.write(c);

// if you've gotten to the end of the line (received a newline

// character) and the line is blank, the http request has ended,

// so you can send a reply

if (c == '\n' && currentLineIsBlank) {

// send a standard http response header

// client.println("HTTP/1.1 200 OK");

// client.println("Content-Type: text/html");

//client.println("Connection: close"); // the connection will be closed after completion of the response

//client.println("Refresh: 1"); // refresh the page automatically every 5 sec

client.println();}

client.println("<!DOCTYPE HTML>");

client.println("<html><head><meta http-equiv=\"refresh\" content=\"1\" />");

client.println("<link rel=\"stylesheet\"

href=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css\">");

client.println("</head><body>");

client.println("<div class='container text-center'>");

if(A=="Street1 is Empty"){

client.println("<button type='button' class='btn btn-success btn-lg'>");

client.println(A);

client.println("</button>");}

else{

client.println("<button type='button' class='btn btn-danger btn-lg'>");

client.println(A+" Go to Street 2");

client.println("</button>");

}

client.println("</div><div class='container text-center'>");

if(B=="Street2 is Empty"){

client.println("<button type='button' class='btn btn-success btn-lg'>");

client.println(B);

client.println("</button>");}

else{

client.println("<button type='button' class='btn btn-danger btn-lg'>");

client.println(B+ " Go to street 1");

client.println("</button>");

}

client.println("</div>");

client.println("</body></html>");

currentLineIsBlank=false;

}

if(!currentLineIsBlank){

break;

}

}

// give the web browser time to receive the data

delay(1);

// close the connection:

client.stop();

Serial.println("client disconnected");

}

// listen for incoming clients

}

# *Testing the project :*

In this phase the project was tested as a one unit and a lot of problems came up and those problems were:

* 1. Connect the wires and problems with connect it to Sensors
  2. regulate and organize each sensor to work with each other
  3. A problem with chose the resistors for tcrt Sensor .
  4. project synchronization and do the work on parallel.
  5. Ethernet shield and receiving request from slaves
  6. A problem with installing grove library and set cursor.

*REFERANCES :*

<https://www.arduino.cc/en/Tutorial/MasterReader>

<https://www.arduino.cc/en/Reference/Wire>

<https://www.arduino.cc/en/Reference/WireBegin>

<https://www.arduino.cc/en/Reference/WireAvailable>

<https://www.arduino.cc/en/Reference/WireOnRequest>

<https://www.arduino.cc/en/Reference/WireRequestFrom>

<https://www.arduino.cc/en/Reference/WireSend>

<https://www.arduino.cc/en/Reference/WireRead>

<https://www.arduino.cc/en/Reference/WireWrite>

<https://www.arduino.cc/en/Tutorial/HelloWorld>

<https://www.arduino.cc/en/Reference/PinMode>

<https://www.arduino.cc/en/Reference/DigitalWrite>

<https://www.arduino.cc/en/Reference/Serial>

<https://www.arduino.cc/en/Serial/Begin>

<http://www.zerokol.com/2012/11/tcrt5000-reflective-optical-sensor.html>

<http://www.vishay.com/optical-sensors/list/product-83760/>

<http://www.vishay.com/docs/83760/tcrt5000.pdf>

[https://www.seeedstudio.com/Grove-LCD-RGB-Backlight-p-1643.html#](https://www.seeedstudio.com/Grove-LCD-RGB-Backlight-p-1643.html)

<http://www.mouser.com/catalog/specsheets/Seeed_104030001.pdf>

<http://wiki.seeed.cc/Grove-LCD_RGB_Backlight/>

<https://community.particle.io/t/solved-port-library-for-grove-lcd-rgb-backlight-display/18553/6>

<https://www.arduino.cc/en/Main/ArduinoEthernetShield>

<https://www.arduino.cc/en/Reference/EthernetBegin>

<https://www.arduino.cc/en/Reference/SPI>

<https://www.arduino.cc/en/Tutorial/WebClient>

<https://www.arduino.cc/en/Reference/EthernetIPAddress>

<https://www.arduino.cc/en/Reference/EthernetServer>

<https://www.arduino.cc/en/Reference/EthernetBegin>

<https://www.arduino.cc/en/Reference/ServerBegin>

<https://www.arduino.cc/en/Reference/EthernetLocalIP>

<https://www.arduino.cc/en/Reference/EthernetClient>

<https://www.arduino.cc/en/Reference/ServerAvailable>

<https://www.arduino.cc/en/Reference/ClientConnected>

<https://www.arduino.cc/en/Reference/ClientAvailable>

<https://www.arduino.cc/en/Reference/ClientRead>

<https://www.arduino.cc/en/Reference/ClientPrintln>

<https://www.arduino.cc/en/Reference/ClientStop>

<https://circuits.io/>