LCD DISPLAY

SENSORS

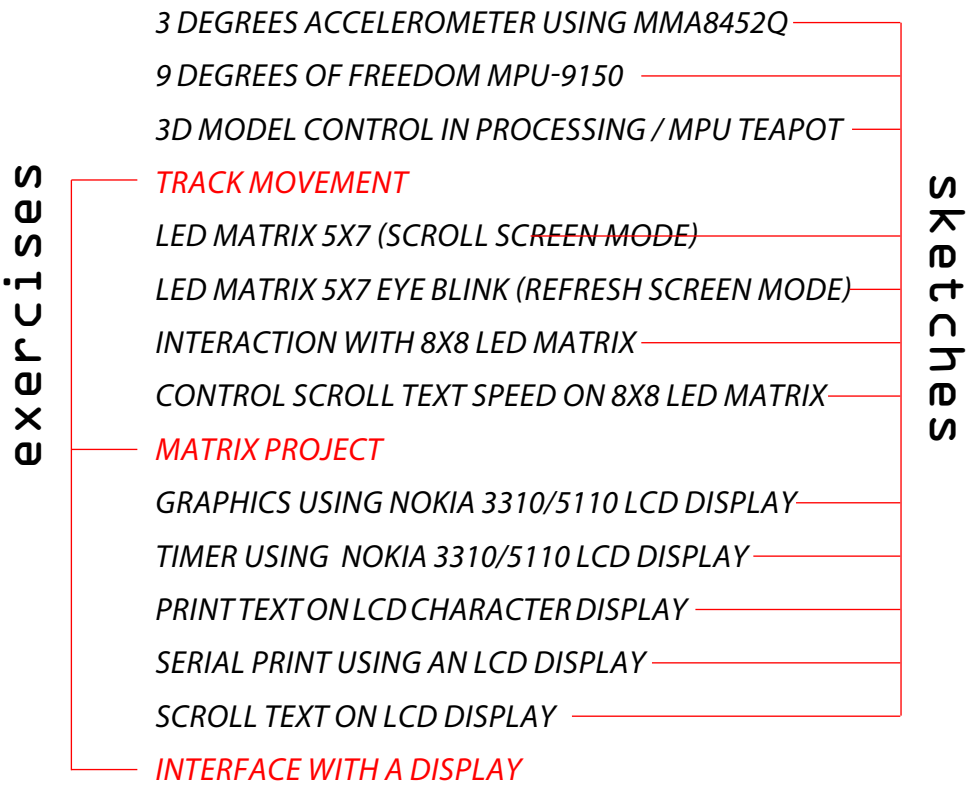

ACTUATORS

LED MATRIX

physical
computing
meals

03

DESSERT


exercises
3 DEGREES ACCELEROMETER USING MMA8452Q
9 DEGREES OF FREEDOM MPU-9150
3D MODEL CONTROL IN PROCESSING / MPU TEAPOT
TRACK MOVEMENT
LED MATRIX 5X7 (SCROLL SCREEN MODE)
LED MATRIX 5X7 EYE BLINK (REFRESH SCREEN MODE)
INTERACTION WITH 8X8 LED MATRIX
CONTROL SCROLL TEXT SPEED ON 8X8 LED MATRIX
MATRIX PROJECT
GRAPHICS USING NOKIA 3310/5110 LCD DISPLAY
TIMER USING NOKIA 3310/5110 LCD DISPLAY
PRINT TEXT ON LCD CHARACTER DISPLAY
SERIAL PRINT USING AN LCD DISPLAY
SCROLL TEXT ON LCD DISPLAY
INTERFACE WITH A DISPLAY
sketches

# 3 Degrees Accelerometer using MMA8452Q

https://www.sparkfun.com/products/10955

This breakout board makes it easy to use the tiny MMA8452Q accelerometer in your project. The MMA8452Q is a smart low-power, three-axis, capacitive micro-machined accelerometer with 12 bits of resolution. This accelerometer is packed with embedded functions with flexible user programmable options, configurable to two interrupt pins. Embedded interrupt functions allow for overall power savings relieving the host processor from continuously polling data.

The MMA8452Q has user selectable full scales of ±2g/±4g/±8g with high pass filtered data as well as non filtered data available real-time. The device can be configured to generate inertial wake-up interrupt signals from any combination of the configurable embedded functions allowing the MMA8452Q to monitor events and remain in a low power mode during periods of inactivity.

This board breaks out the ground, power, I2C and two external interrupt pins.

Not sure which accelerometer is right for you? Our Accelerometer and Gyro Buying Guide might help!

Various type of Accelerometers:
https://www.sparkfun.com/pages/accel_gyro_guide

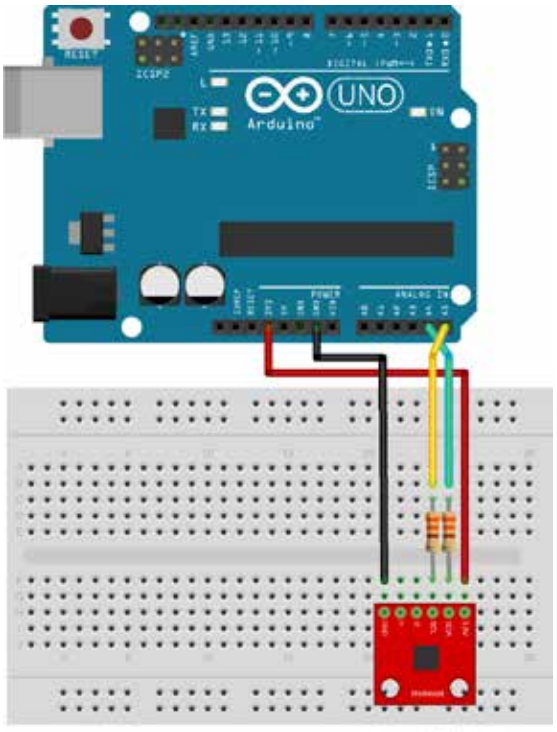
*HARDWARE REQUIRED*

Arduino Board
(1) MMA8452 Accelerometer
2 330 Ohm Resistors

*CIRCUIT*

**Hardware setup:**
MMA8452 Breakout ---------- Arduino
3.3V ---------- 3.3V
SDA ----------^^(330 Resistor)^^-------- A4
SCL ----------^^(330 Resistor)^^-------- A5
GND ---------- GND

*IMAGE*



*CODE*

```
/*
 MMA8452Q Basic Example Code
 Nathan Seidle
 SparkFun Electronics
 November 5, 2012

 License: This code is public domain but you buy me a beer if you use this and we meet someday (Beerware license).

 This example code shows how to read the X/Y/Z accelerations and basic functions of the MMA5842. It leaves out
 all the neat features this IC is capable of (tap, orientation, and inerrupts) and just displays X/Y/Z. See  the advanced example code to see more features.

 Hardware setup:
 MMA8452 Breakout ------------ Arduino
 3.3V ---------------------- 3.3V
 SDA --------^^(330)^^------- A4
 SCL --------^^(330)^^------- A5
 GND ---------------------- GND

 The MMA8452 is 3.3V so we recommend using 330 or 1k resistors between a 5V Arduino and the MMA8452 breakout.

 The MMA8452 has built in pull-up resistors for I2C so you do not need additional pull-ups.
 */

#include <Wire.h> // Used for I2C

// The SparkFun breakout board defaults to 1,
// set to 0 if SA0 jumper on the bottom of the board is set

#define MMA8452_ADDRESS 0x1D  // 0x1D if SA0 is high, 0x1C if low

//Define a few of the registers that we will be accessing on the MMA8452
#define OUT_X_MSB 0x01
#define XYZ_DATA_CFG  0x0E
#define WHO_AM_I   0x0D
#define CTRL_REG1  0x2A
#define GSCALE 2 // Sets full-scale range to +/-2, 4, or 8g. Used to calc real g values.

void setup()
{
  Serial.begin(57600);
  Serial.println("MMA8452 Basic Example");

  Wire.begin(); //Join the bus as a master
  initMMA8452(); //Test and intialize the MMA8452
}

void loop()
{
  int accelCount[3];  // Stores the 12-bit signed value
  readAccelData(accelCount);  // Read the x/y/z adc values

  // Now we'll calculate the accleration value into actual g's
  float accelG[3];  // Stores the real accel value in g's
  for (int i = 0 ; i < 3 ; i++)
  {
    accelG[i] = (float) accelCount[i] / ((1<<12)/(2*GSCALE));
// get actual g value, this depends on scale being set
  }  for (int i = 0 ; i < 3 ; i++)   // Print out values
  {
    Serial.print(accelG[i], 4);  // Print g values
    Serial.print("\t");  // tabs in between axes
  }=
  Serial.println();
  delay(10);  // Delay here for visibility
}

void readAccelData(int *destination)
{
  byte rawData[6];  // x/y/z accel register data stored here
  readRegisters(OUT_X_MSB, 6, rawData);  // Read the six raw data registers into data array
    for (int i = 0; i < 3 ; i++) // Loop to calculate 12-bit ADC and g value for each axis
  {
    int gCount = (rawData[i*2] << 8) | rawData[(i*2)+1];  //Combine the two 8 bit registers into one 12-bit number
    gCount >>= 4; //The registers are left align, here we right align the 12-bit integer

    // If the number is negative, we have to make it so manually (no 12-bit data type)

    if (rawData[i*2] > 0x7F)
    {
      gCount = ~gCount + 1;
      gCount *= -1;  // Transform into negative 2's complement #
    }
    destination[i] = gCount; //Record this gCount into the 3 int array
  }
}
```

```
// Initialize the MMA8452 registers
// See the many application notes for more info on setting all of these registers:
// http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MMA8452Q

void initMMA8452()
{
  byte c = readRegister(WHO_AM_I);  // Read WHO_AM_I register
  if (c == 0x2A) // WHO_AM_I should always be 0x2A
  {
    Serial.println("MMA8452Q is online...");
  }
  else
  {
    Serial.print("Could not connect to MMA8452Q: 0x");
    Serial.println(c, HEX);
    while(1) ; // Loop forever if communication doesn't happen
  }

  MMA8452Standby();  // Must be in standby to change registers

  // Set up the full scale range to 2, 4, or 8g.
  byte fsr = GSCALE;
  if(fsr > 8) fsr = 8; //Easy error check
  fsr >>= 2; // Neat trick, see page 22. 00 = 2G, 01 = 4A, 10 = 8G
  writeRegister(XYZ_DATA_CFG, fsr);

  //The default data rate is 800Hz and we don't modify it in this example code

  MMA8452Active();  // Set to active to start reading
}

// Sets the MMA8452 to standby mode. It must be in standby to change most register settings
void MMA8452Standby()
{
  byte c = readRegister(CTRL_REG1);
  writeRegister(CTRL_REG1, c & ~(0x01)); //Clear the active bit to go into standby
}

// Sets the MMA8452 to active mode. Needs to be in this mode to output data
void MMA8452Active()
{
  byte c = readRegister(CTRL_REG1);
  writeRegister(CTRL_REG1, c | 0x01); //Set the active bit to begin detection
}

// Read bytesToRead sequentially, starting at addressToRead into the dest byte array
void readRegisters(byte addressToRead, int bytesToRead, byte * dest)
{
  Wire.beginTransmission(MMA8452_ADDRESS);
  Wire.write(addressToRead);
  Wire.endTransmission(false); //endTransmission but keep the connection active

  Wire.requestFrom(MMA8452_ADDRESS, bytesToRead); //Ask for bytes, once done, bus is released by default

  while(Wire.available() < bytesToRead); //Hang out until we get the # of bytes we expect

  for(int x = 0 ; x < bytesToRead ; x++)
    dest[x] = Wire.read();
}

// Read a single byte from addressToRead and return it as a byte
byte readRegister(byte addressToRead)
{
  Wire.beginTransmission(MMA8452_ADDRESS);
```

```
  Wire.write(addressToRead);
  Wire.endTransmission(false); //endTransmission but keep the connection active

  Wire.requestFrom(MMA8452_ADDRESS, 1); //Ask for 1 byte, once done, bus is released by default

  while(!Wire.available()) ; //Wait for the data to come back
  return Wire.read(); //Return this one byte
}

// Writes a single byte (dataToWrite) into addressToWrite
void writeRegister(byte addressToWrite, byte dataToWrite)
{
  Wire.beginTransmission(MMA8452_ADDRESS);
  Wire.write(addressToWrite);
  Wire.write(dataToWrite);
  Wire.endTransmission(); //Stop transmitting
}
```

# 9 Degrees of Freedom MPU-9150

https://www.sparkfun.com/products/11486
https://github.com/Pansenti/MPU9150Lib

The MPU-9150 is the world's first 9-axis MotionTracking device designed for the low power, low cost, and high performance requirements of consumer electronics equipment including smartphones, tablets and wearable sensors. And guess what? You get to play with it.

This breakout board makes it easy to prototype with the InvenSense MPU-9150 by breaking out all the pins you need to standard 0.1" spaced headers. The board also provides I2C pullup resistors and a solder jumper to switch the I2C address of the device. The MPU-9150 is a System in Package (SiP) that combines two chips: the MPU-6050, which contains a 3-axis gyroscope, 3-axis accelerometer, and an on-board Digital Motion Processor (DMP) capable of processing complex 9-axis MotionFusion algorithms; and the AK8975, a 3-axis digital compass. The part's integrated 9-axis MotionFusion algorithms access all internal sensors to gather a full set of sensor data. The part is offered in a 4x4x1mm LGA package and is upgrade-compatible with the MPU-6050 integrated 6-axis MotionTracking device, providing a simple upgrade path and making it easy to fit on space constrained boards.

*HARDWARE REQUIRED*

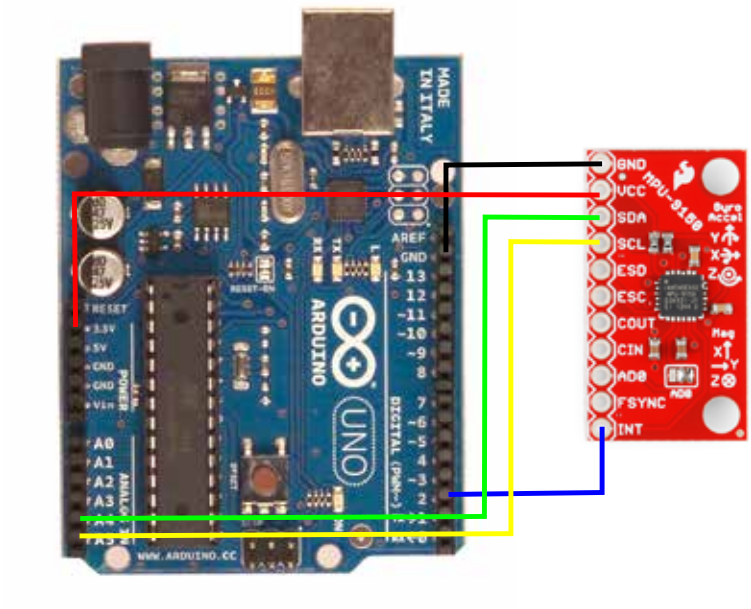
Arduino Board
(1) MPU-9150 Motion Tracking Device

*CIRCUIT*

MPU-9150 GRD > Arduino Ground
MPU-9150 VCC > Arduino 3.3 VOLT
MPU-9150 SDA > Arduino pin A4
MPU-9150 SCL > Arduino pin A5
MPU-9150 INT > Arduino pin 2

## Futures

Digital-output 9-axis MotionFusion data in rotation matrix, quaternion, Euler Angle, or raw data format
Tri-Axis angular rate sensor (gyro) with a sensitivity up to 131 LSBs/dps and a full-scale range of ±250, ±500, ±1000, and ±2000dps
Tri-Axis accelerometer with a programmable full scale range of ±2g, ±4g, ±8g and ±16g
Tri-axis compass with a full scale range of ±1200µT
Reduced settling effects and sensor drift by elimination of board-level cross-axis alignment errors between accelerometer, gyroscope, and compass
VDD Supply voltage range of 2.4V–3.46V; VLOGIC of 1.8V±5% or VDD
Gyro operating current: 3.6mA (full power, gyro at all rates)
Gyro + Accel operating current: 3.8mA (full power, gyro at all rates, accel at 1kHz sample rate)
Gyro + Accel + Compass + DMP operating current: 4.25mA (full power, gyro at all rates, accel at 1kHz sample rate, compass at 8Hz rate)
Accel low power mode operating current: 10uA at 1Hz, 20uA at 5Hz, 70uA at 20Hz, 140uA at 40Hz
Full Chip Idle Mode Supply Current: 8µA
400kHz Fast Mode $I^2C$ serial host interface
On-chip timing generator with ±1% frequency variation over full temperature range
10,000g shock tolerant
I2C Pullup Resistors populated on board.
All Pins Broken Out to Standard 0.1» Spaced Headers
Solder Jumper for Switching LSB of I2C Address




*IMAGE*

*CODE*

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU9150
// 1/4/2013 original by Jeff Rowberg <jeff@rowberg.net> at https://github.com/jrowberg/i2cdevlib
//  modified by Aaron Weiss <aaron@sparkfun.com>
//
// Changelog:
//     2011-10-07 - initial release
//     2013-1-4 - added raw magnetometer output

/* ============================================
I2Cdev device library code is placed under the MIT license

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the «Software»), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED «AS IS», WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
=============================================== */

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation is used in I2Cdev.h
#include "Wire.h"

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050.h"

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 accelgyro;

int16_t ax, ay, az;
int16_t gx, gy, gz;
int16_t mx, my, mz;

#define LED_PIN 13
bool blinkState = false;

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    Wire.begin();

    // initialize serial communication
    // (38400 chosen because it works as well at 8MHz as it does at 16MHz, but
    // it's really up to you depending on your project)
    Serial.begin(38400);

    // initialize device
    Serial.println("Initializing I2C devices...");
    accelgyro.initialize();

    // verify connection
    Serial.println("Testing device connections...");
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection failed");

    // configure Arduino LED for
    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    // read raw accel/gyro measurements from device
    accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);

    // these methods (and a few others) are also available
    //accelgyro.getAcceleration(&ax, &ay, &az);
    //accelgyro.getRotation(&gx, &gy, &gz);

    // display tab-separated accel/gyro x/y/z values
    Serial.print("a/g/m:\t");
    Serial.print(ax); Serial.print("\t"); // \t is used to print TAB
    Serial.print(ay); Serial.print("\t");
    Serial.print(az); Serial.print("\t");
    Serial.print(gx); Serial.print("\t");
    Serial.print(gy); Serial.print("\t");
    Serial.print(gz); Serial.print("\t");
    Serial.print(mx); Serial.print("\t");
    Serial.print(my); Serial.print("\t");
    Serial.println(mz);

    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
}
```

# 3D model control in Processing / MPU Teapot

http://www.youtube.com/watch?v=74xL-VcRyjQ

Arduino Board
(1) MPU-9150 Motion Tracking Device

MPU-9150 GRD > Arduino Ground
MPU-9150 VCC > Arduino 3.3 VOLT
MPU-9150 SDA > Arduino pin A4
MPU-9150 SCL > Arduino pin A5
MPU-9150 INT > Arduino pin 2

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using
DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/
i2cdevlib

// Changelog:
//     2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//     2012-06-20 - improved FIFO overflow handling and simplified read process
//     2012-06-19 - completely rearranged DMP initialization code and simplification
//     2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//     2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//     2012-06-05 - add gravity-compensated initial reference frame acceleration output
//     - add 3D math helper file to DMP6 example sketch
//     - add Euler output and Yaw/Pitch/Roll output formats
//     2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//     2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//     2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN
THE SOFTWARE.
===============================================
*/
```

*ARDUINO CODE*

```
// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation is used in I2Cdev.h
#include "Wire.h"

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// class default I2C address is 0x68 specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;

/* =========================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch depends on the MPU-6050's INT pin being connected to the Arduino's external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is digital I/O pin 2.
 * =========================================*/

/* =========================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error when using Serial.write(buf, len). The Teapot output uses this method. The solution requires a modification to the Arduino USBAPI.h file, which is fortunately simple, but annoying. This will be fixed in the next IDE release. For more info, see these links:
   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 * =========================================*/

// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
#define OUTPUT_TEAPOT

#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;
```

*ARDUINO CODE*

```
// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0
= error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z] quaternion container
VectorInt16 aa;         // [x, y, z]    accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]  gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]  world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]  gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];       // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r',
'\n' };


// ===============================================
// ===           INTERRUPT DETECTION ROUTINE           ===
// ===============================================

// indicates whether MPU interrupt pin has gone high
volatile bool mpuInterrupt = false;

void dmpDataReady() {
    mpuInterrupt = true;
}

// ===============================================
// ===                INITIAL SETUP                ===
// ===============================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    Wire.begin();

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
    // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful")
: F("MPU6050 connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and
demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();
```

```
    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.println(F("Enabling interrupt detection (Arduino
external interrupt 0)..."));
        attachInterrupt(0, dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's
okay to use it
        Serial.println(F("DMP ready! Waiting for first inter-
rupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}

// ================================================
// ===                    MAIN PROGRAM LOOP                    ===
// ================================================

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // other program behavior stuff here
        // if you are really paranoid you can frequently test in between other
        // stuff to see if mpuInterrupt is true, and if so, "break;" from the
        // while() loop to immediately process the MPU data
    }

    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();


    // check for overflow (this should never happen unless our code is
too inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));


// otherwise, check for DMP data ready interrupt (this should happen
frequently)
    } else if (mpuIntStatus & 0x02) {
        // wait for correct available data length, should be a VERY short
wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFO-
Count();
```

```
        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an
interrupt)
        fifoCount -= packetSize;

        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("\t");
            Serial.print(q.x);
            Serial.print("\t");
            Serial.print(q.y);
            Serial.print("\t");
            Serial.println(q.z);
        #endif

        #ifdef OUTPUT_READABLE_EULER
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetEuler(euler, &q);
            Serial.print("euler\t");
            Serial.print(euler[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(euler[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(euler[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_REALACCEL
            // display real acceleration, adjusted to remove gravity
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            Serial.print("areal\t");
            Serial.print(aaReal.x);
            Serial.print("\t");
            Serial.print(aaReal.y);
            Serial.print("\t");
            Serial.println(aaReal.z);
        #endif

        #ifdef OUTPUT_READABLE_WORLDACCEL
            // display initial world-frame acceleration, adjusted to remove
gravity

// and rotated based on known orientation from quaternion
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal,
&q);
```

```
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo
format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on
purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

## *PROCESSING CODE*

```
// I2C device class (I2Cdev) demonstration Processing sketch for MPU6050 DMP output
// 6/20/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//     2012-06-20 - initial release

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
============================================
*/
```

```
import processing.serial.*;
import processing.opengl.*;
import toxi.geom.*;
import toxi.processing.*;

// NOTE: requires ToxicLibs to be installed in order to run properly.
// 1. Download from http://toxiclibs.org/downloads
// 2. Extract into [userdir]/Processing/libraries
//    (location may be different on Mac/Linux)
// 3. Run and bask in awesomeness

ToxiclibsSupport gfx;

Serial port;                          // The serial port
char[] teapotPacket = new char[14];  // InvenSense Teapot packet
int serialCount = 0;                 // current packet byte position
int aligned = 0;
int interval = 0;

float[] q = new float[4];
Quaternion quat = new Quaternion(1, 0, 0, 0);

float[] gravity = new float[3];
float[] euler = new float[3];
float[] ypr = new float[3];

void setup() {
    // 300px square viewport using OpenGL rendering
    size(300, 300, OPENGL);
    gfx = new ToxiclibsSupport(this);

    // setup lights and antialiasing
    lights();
    smooth();

    // display serial port list for debugging/clarity
    println(Serial.list());

    // get the first available port (use EITHER this OR the specific port code below)
    String portName = Serial.list()[2];

    // get a specific serial port (use EITHER this OR the first-available code above)
    //String portName = "COM4";

    // open the serial port
    port = new Serial(this, portName, 115200);

    // send single character to trigger DMP init/start
    // (expected by MPU6050_DMP6 example Arduino sketch)
    port.write('r');
}

void draw() {
    if (millis() - interval > 1000) {
        // resend single character to trigger DMP init/start
        // in case the MPU is halted/reset while applet is running
        port.write('r');
        interval = millis();
    }

    // black background
    background(0);
    // translate everything to the middle of the viewport
    pushMatrix();
    translate(width / 2, height / 2);

    // 3-step rotation from yaw/pitch/roll angles (gimbal lock!)
    // ...and other weirdness I haven't figured out yet
    //rotateY(-ypr[0]);
    //rotateZ(-ypr[1]);
    //rotateX(-ypr[2]);
```

*PROCESSING CODE*

```
    // toxiclibs direct angle/axis rotation from quaternion (NO gimbal lock!)
    // (axis order [1, 3, 2] and inversion [-1, +1, +1] is a consequence of
    // different coordinate system orientation assumptions between Processing
    // and InvenSense DMP)
    float[] axis = quat.toAxisAngle();
    rotate(axis[0], -axis[1], axis[3], axis[2]);

    // draw main body in red
    fill(255, 0, 0, 200);
    box(10, 10, 200);

    // draw front-facing tip in blue
    fill(0, 0, 255, 200);
    pushMatrix();
    translate(0, 0, -120);
    rotateX(PI/2);
    drawCylinder(0, 20, 20, 8);
    popMatrix();

    // draw wings and tail fin in green
    fill(0, 255, 0, 200);
    beginShape(TRIANGLES);
    vertex(-100,  2, 30); vertex(0,  2, -80); vertex(100,  2, 30);
 // wing top layer
    vertex(-100, -2, 30); vertex(0, -2, -80); vertex(100, -2, 30);
// wing bottom layer
    vertex(-2, 0, 98); vertex(-2, -30, 98); vertex(-2, 0, 70);
// tail left layer
    vertex( 2, 0, 98); vertex( 2, -30, 98); vertex( 2, 0, 70);
// tail right layer
    endShape();
    beginShape(QUADS);
    vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(  0, -2, -80); vertex(
0, 2, -80);
    vertex( 100, 2, 30); vertex( 100, -2, 30); vertex(  0, -2, -80); vertex(
0, 2, -80);
    vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(100,-2, 30); vertex(
100, 2,  30);
    vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2, -30, 98); vertex (-2,
-30, 98);
    vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2,   0, 70); vertex(-2,
0, 70);
    vertex(-2, -30, 98); vertex(2, -30, 98); vertex(2,   0, 70); vertex(-2,
0, 70);
    endShape();

    popMatrix();
}

void serialEvent(Serial port) {
    interval = millis();
    while (port.available() > 0) {
        int ch = port.read();
        print((char)ch);
        if (aligned < 4) {
            // make sure we are properly aligned on a 14-byte packet
            if (serialCount == 0) {
                if (ch == '$') aligned++; else aligned = 0;
            } else if (serialCount == 1) {
                if (ch == 2) aligned++; else aligned = 0;
            } else if (serialCount == 12) {
                if (ch == '\r') aligned++; else aligned = 0;
            } else if (serialCount == 13) {
                if (ch == '\n') aligned++; else aligned = 0;
            }
            //println(ch + " " + aligned + " " + serialCount);
            serialCount++;
            if (serialCount == 14) serialCount = 0;
        } else {
```

*PROCESSING CODE*

```
            if (serialCount > 0 || ch == '$') {
                teapotPacket[serialCount++] = (char)ch;
                if (serialCount == 14) {
                    serialCount = 0; // restart packet byte position

                    // get quaternion from data packet
                    q[0] = ((teapotPacket[2] << 8) | teapotPack-
et[3]) / 16384.0f;
                    q[1] = ((teapotPacket[4] << 8) | teapotPack-
et[5]) / 16384.0f;
                    q[2] = ((teapotPacket[6] << 8) | teapotPack-
et[7]) / 16384.0f;
                    q[3] = ((teapotPacket[8] << 8) | teapotPack-
et[9]) / 16384.0f;
                    for (int i = 0; i < 4; i++) if (q[i] >= 2) q[i] = -4
+ q[i];

                    // set our toxilibs quaternion to new data
                    quat.set(q[0], q[1], q[2], q[3]);

                    /*
                   // below calculations unnecessary for orientation only
using toxilibs

                    // calculate gravity vector
                    gravity[0] = 2 * (q[1]*q[3] - q[0]*q[2]);
                   gravity[1] = 2 * (q[0]*q[1] + q[2]*q[3]);
                   gravity[2] = q[0]*q[0] - q[1]*q[1] - q[2]*q[2] +
q[3]*q[3];

                   // calculate Euler angles
                   euler[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3],
2*q[0]*q[0] + 2*q[1]*q[1] - 1);
                   euler[1] = -asin(2*q[1]*q[3] + 2*q[0]*q[2]);
                   euler[2] = atan2(2*q[2]*q[3] - 2*q[0]*q[1],
2*q[0]*q[0] + 2*q[3]*q[3] - 1);

                   // calculate yaw/pitch/roll angles
                   ypr[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0]
+ 2*q[1]*q[1] - 1);
                   ypr[1] = atan(gravity[0] / sqrt(gravity[1]*gravity[1] +
gravity[2]*gravity[2]));
                   ypr[2] = atan(gravity[1] / sqrt(gravity[0]*gravity[0] +
gravity[2]*gravity[2]));

                   // output various components for debugging
                   //println("q:\t" + round(q[0]*100.0f)/100.0f + "\t" +
round(q[1]*100.0f)/100.0f + "\t" + round(q[2]*100.0f)/100.0f + "\t"
+ round(q[3]*100.0f)/100.0f);
                   //println("euler:\t" + euler[0]*180.0f/PI + "\t" +
euler[1]*180.0f/PI + "\t" + euler[2]*180.0f/PI);
                   //println("ypr:\t" + ypr[0]*180.0f/PI + "\t" +
ypr[1]*180.0f/PI + "\t" + ypr[2]*180.0f/PI);
                   */
                }
            }
        }
    }
}

void drawCylinder(float topRadius, float bottomRadius, float
tall, int sides) {
    float angle = 0;
    float angleIncrement = TWO_PI / sides;
    beginShape(QUAD_STRIP);
    for (int i = 0; i < sides + 1; ++i) {
        vertex(topRadius*cos(angle), 0, topRadius*sin(angle));
        vertex(bottomRadius*cos(angle), tall,
bottomRadius*sin(angle));
        angle += angleIncrement;
    }
    endShape();
```

*PROCESSING CODE*

```
    // If it is not a cone, draw the circular top cap
    if (topRadius != 0) {
        angle = 0;
        beginShape(TRIANGLE_FAN);

        // Center point
        vertex(0, 0, 0);
        for (int i = 0; i < sides + 1; i++) {
            vertex(topRadius * cos(angle), 0, topRadius *
sin(angle));
            angle += angleIncrement;
        }
        endShape();
    }

    // If it is not a cone, draw the circular bottom cap
    if (bottomRadius != 0) {
        angle = 0;
        beginShape(TRIANGLE_FAN);

        // Center point
        vertex(0, tall, 0);
        for (int i = 0; i < sides + 1; i++) {
            vertex(bottomRadius * cos(angle), tall, bottomRadius
* sin(angle));
            angle += angleIncrement;
        }
        endShape();
    }
}
```

## Track Movement

*EXERCISE*

Use a accelerometer on your project. Try to embed the navigation data to a mechanical project .

*HARDWARE REQUIRED*

Arduino Board
Accelerometer
Servo motor

*SCHEMATIC*

# LED matrix 5X7 (Scroll Screen Mode)

www.hanez.org
http://hanez.org/arduino-ta07-11-5x7-dot-matrix-display.html#Code

CONNECTION: From LED Matrix to Arduino Pins.
1 to 1, 2 to 2, 3 to 3, 4 to 4, 5 to 5, 6 to 6, (one side)
7 to 7, 8 to 8, 9 to 9, 10 to 10, 11 to 11, 12 to 12 (other side)

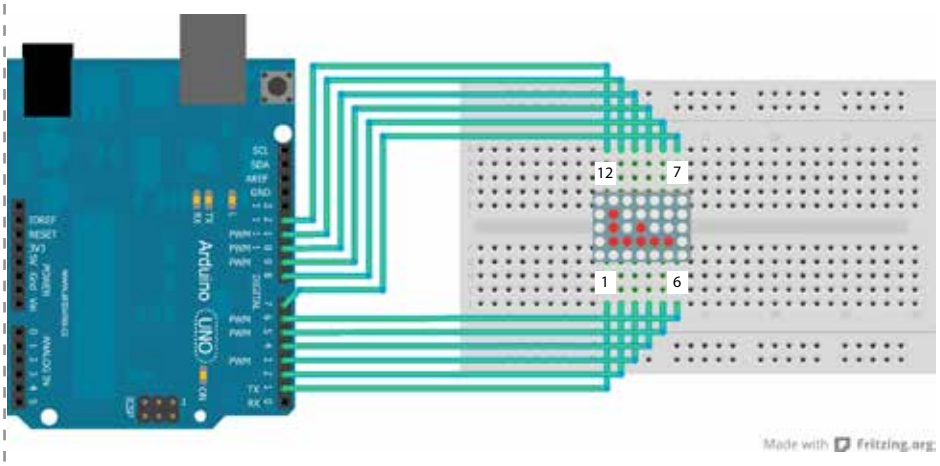
*HARDWARE REQUIRED*

Arduino Board
5X7 LED Matrix (Kingbright TA07-11EWA) Common Anode or Cathode

*CIRCUIT*

**This projects uses the TimeOne Library.**
The character set is based on MatrixFont3X5 library . It produce a scroll text and some blink effects.
**IMPORT NOTICE**
LED displays are often packaged as matrices of LEDs arranged in rows of common anodes and columns of common cathodes, or the reverse.
FOR COMMON ANODE LED MATRIX SET THE Current varible equal to 1
FOR COMMON CATHODE LED MATRIX SET THE Current varible equal to 0

*IMAGE*




*CODE*

```
#include <TimerOne.h>
#include <MatrixFonts3x5.h>

#define COLS 5
#define ROWS 7
#define PINS 13

#define MATRIX1 { \
  {1,0,1,0,1,0,1}, \
  {0,1,0,1,0,1,0}, \
  {1,0,1,0,1,0,1}, \
  {0,1,0,1,0,1,0}, \
  {1,0,1,0,1,0,1} \
}

#define MATRIX2 { \
  {0,1,0,1,0,1,0}, \
  {1,0,1,0,1,0,1}, \
  {0,1,0,1,0,1,0}, \
  {1,0,1,0,1,0,1}, \
  {0,1,0,1,0,1,0} \
}
```

```
// For LED Matrix Common Anode choose Current=0;
// For LED Matrix Common Cathode choose Current=1;
int Current=1;

byte col = 0;
byte leds[COLS][ROWS];

// pin[xx] on led matrix connected to nn on Arduino (-1 is dummy to make array
start at pos 1)
int pins[PINS]= {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

// col[xx] of leds = pin yy on led matrix
int cols[COLS] = {pins[1], pins[3], pins[10], pins[7], pins[8]};

// row[xx] of leds = pin yy on led matrix
int rows[ROWS] = {pins[12], pins[11], pins[2], pins[9], pins[4], pins[5],
pins[6]};

const int numPatterns = 14;
byte patterns[numPatterns][COLS][ROWS] = {H,A,L,L,O,DASH,W,E,L,T,D
OT,DOT,DOT,SPACE};

int pattern = 0;

boolean normal;


void setup() {
  //  Switch the current flow in Rows and Columns
    if (Current==1) {
     normal=1;
    } else {
   normal=0;
}

  // sets the pins as output
  for (int i = 0; i < PINS; i++) { pinMode(pins[i], OUTPUT); }

  // set up cols
  for (int i = 1; i <= COLS; i++) { digitalWrite(cols[i - 1], normal); }

  //  and rows
  for (int i = 1; i <= ROWS; i++) { digitalWrite(rows[i - 1], !normal); }

  blink();
  five2one();

  clearLeds();

  Timer1.initialize(2000);         // initialize timer1, and set a 1/2 second period
  Timer1.attachInterrupt(display);  // attaches display() as a timer overflow
interrupt

  setPattern(pattern);
}

void loop() {
 pattern = ++pattern % numPatterns;
 slidePattern(pattern, 180);
}

void display() { // Interrupt routine
  digitalWrite(cols[col], normal);  // Turn whole previous column off
  col++;
  if (col == 5) {
    col = 0;
  }
  for (int row = 0; row < 7; row++) {
    if (leds[col][(ROWS - 1) - row] == 1) {
      digitalWrite(rows[row], normal);  // Turn on this led
    }  else {
```

```
    digitalWrite(rows[row], !normal); // Turn off this led
   }
  }
  digitalWrite(cols[col], !normal); // Turn whole column on at once
}

void slidePattern(int pattern, int del) {
  for (int l = 0; l < COLS; l++) {
    for (int i = 0; i < (COLS - 1); i++) {
      for (int j = 0; j < ROWS; j++) {
        leds[i][j] = leds[i + 1][j];
      }
    }
    for (int j = 0; j < ROWS; j++) {
      leds[4][j] = patterns[pattern][0 + l][j];
    }
    delay(del);
  }
}

void clearLeds() {
  // Clear display array
  for (int i = 0; i < COLS; i++) {
    for (int j = 0; j < ROWS; j++) {
      leds[i][j] = 0;
    }
  }
}

void setPattern(int pattern) {
  for (int i = 0; i < COLS; i++) {
    for (int j = 0; j < ROWS; j++) {
      leds[i][j] = patterns[pattern][i][j];
    }
  }
}

void blink() {
  for(int i = 0; i < 12; i++) {
    if(i % 2 != 0) {
      digitalWrite(cols[0], !normal);
      digitalWrite(cols[1], normal);
      digitalWrite(cols[2], normal);
      digitalWrite(cols[3], normal);
      digitalWrite(cols[4], normal);
      digitalWrite(rows[0], normal);
      digitalWrite(rows[1], !normal);
      digitalWrite(rows[2], !normal);
      digitalWrite(rows[3], !normal);
      digitalWrite(rows[4], !normal);
      digitalWrite(rows[5], !normal);
      digitalWrite(rows[6], !normal);
    } else {
      digitalWrite(cols[0], normal);
      digitalWrite(cols[1], normal);
      digitalWrite(cols[2], normal);
      digitalWrite(cols[3], normal);
      digitalWrite(cols[4], normal);
      digitalWrite(rows[0], !normal);
      digitalWrite(rows[1], !normal);
      digitalWrite(rows[2], !normal);
      digitalWrite(rows[3], !normal);
      digitalWrite(rows[4], !normal);
      digitalWrite(rows[5], !normal);
      digitalWrite(rows[6], !normal);
    }
    delay(100);
  }
}
```

*CODE*

```
void five2one() {
  digitalWrite(cols[0], !normal);
  digitalWrite(cols[1], normal);
  digitalWrite(cols[2], normal);
  digitalWrite(cols[3], normal);
  digitalWrite(cols[4], normal);
  digitalWrite(rows[0], normal);
  digitalWrite(rows[1], !normal);
  digitalWrite(rows[2], !normal);
  digitalWrite(rows[3], !normal);
  digitalWrite(rows[4], !normal);
  digitalWrite(rows[5], !normal);
  digitalWrite(rows[6], !normal);
  delay(100);
  digitalWrite(cols[0], normal);
  digitalWrite(cols[1], !normal);
  digitalWrite(cols[2], normal);
  digitalWrite(cols[3], normal);
  digitalWrite(cols[4], normal);
  digitalWrite(rows[0], normal);
  digitalWrite(rows[1], !normal);
  digitalWrite(rows[2], !normal);
  digitalWrite(rows[3], !normal);
  digitalWrite(rows[4], !normal);
  digitalWrite(rows[5], !normal);
  digitalWrite(rows[6], !normal);
  delay(100);
  digitalWrite(cols[0], normal);
  digitalWrite(cols[1], normal);
  digitalWrite(cols[2], !normal);
  digitalWrite(cols[3], normal);
  digitalWrite(cols[4], normal);
  digitalWrite(rows[0], normal);
  digitalWrite(rows[1], !normal);
  digitalWrite(rows[2], !normal);
  digitalWrite(rows[3], !normal);
  digitalWrite(rows[4], !normal);
  digitalWrite(rows[5], !normal);
  digitalWrite(rows[6], !normal);
  delay(100);
  digitalWrite(cols[0], normal);
  digitalWrite(cols[1], normal);
  digitalWrite(cols[2], normal);
  digitalWrite(cols[3], !normal);
  digitalWrite(cols[4], normal);
  digitalWrite(rows[0], normal);
  digitalWrite(rows[1], !normal);
  digitalWrite(rows[2], !normal);
  digitalWrite(rows[3], !normal);
  digitalWrite(rows[4], !normal);
  digitalWrite(rows[5], !normal);
  digitalWrite(rows[6], !normal);
  delay(100);
  digitalWrite(cols[0], normal);
  digitalWrite(cols[1], normal);
  digitalWrite(cols[2], normal);
  digitalWrite(cols[3], normal);
  digitalWrite(cols[4], !normal);
  digitalWrite(rows[0], normal);
  digitalWrite(rows[1], !normal);
  digitalWrite(rows[2], !normal);
  digitalWrite(rows[3], !normal);
  digitalWrite(rows[4], !normal);
  digitalWrite(rows[5], !normal);
  digitalWrite(rows[6], !normal);
  delay(100l);
}
```

# LED matrix 5X7 Eye Blink (Refresh Screen Mode)

www.hanez.org
http://hanez.org/arduino-ta07-11-5x7-dot-matrix-display.html#Code

CONNECTION: From LED Matrix to Arduino Pins.
1 to 1, 2 to 2, 3 to 3, 4 to 4, 5 to 5, 6 to 6, (one side)
7 to 7, 8 to 8, 9 to 9, 10 to 10, 11 to 11, 12 to 12 (other side)

*HARDWARE REQUIRED*

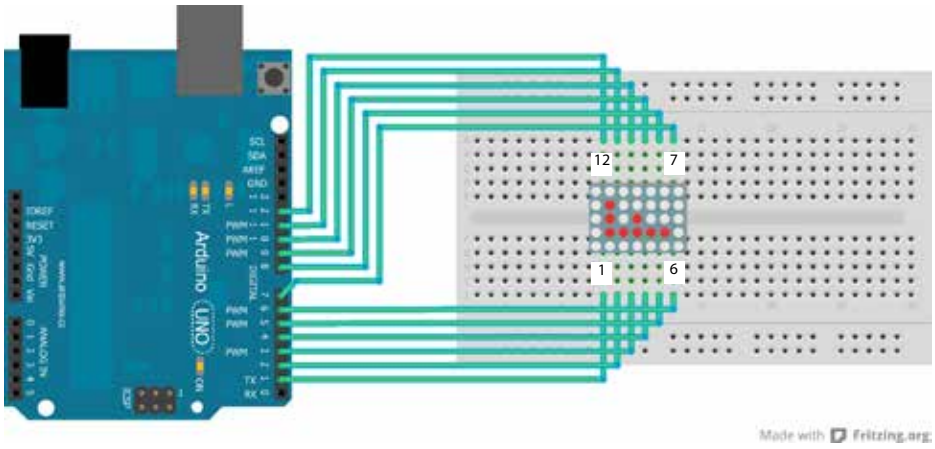
Arduino Board
5X7 LED Matrix (Kingbright TA07-11EWA) Common Anode or Cathode

*CIRCUIT*

**This projects uses the FrequensyTimer2 Library.**

**IMPORT NOTICE**
LED displays are often packaged as matrices of LEDs arranged in rows of common anodes and columns of common cathodes, or the reverse.
FOR COMMON ANODE LED MATRIX SET THE Current varible equal to 1
FOR COMMON CATHODE LED MATRIX SET THE Current varible equal to 0

*IMAGE*




*CODE*

```
// FROM 5X7 LED MATRIX GOES TO ARDUINO PINS
// 1>2, 2>9, 3>3, 4>11, 5>12, 6>13, 7>7, 8>8, 9>4, 10>10, 11>6, 12>5

#include <FrequencyTimer2.h>

#define A { \
{1, 0, 0, 0, 0}, \
{1, 1, 1, 1, 0}, \
{1, 0, 0, 0, 1}, \
{1, 0, 0, 0, 1}, \
{1, 0, 0, 0, 1}, \
{1, 1, 1, 1, 0}, \
{1, 0, 0, 0, 0} \
}
```

*CODE*

```
* Source: http://www.arduino.cc/playground/
Main/DirectDriveLEDMatrix
*
* Modifications for 5x7 LED Matrix element
* by Stefan Wolfrum in July 2012.
* ----------------------------------------
*
* Show messages on an 5x7 led matrix,
* scrolling from right to left.
*
* Uses FrequencyTimer2 library to
* constantly run an interrupt routine
* at a specified frequency. This
* refreshes the display without the
* main loop having to do anything.
*
*/
```

```
#define B { \
{0, 1, 0, 0, 0}, \
{1, 1, 1, 1, 0}, \
{1, 0, 0, 0, 1}, \
{1, 0, 0, 0, 1}, \
{1, 0, 0, 0, 1}, \
{1, 1, 1, 1, 0}, \
{0, 1, 0, 0, 0} \
}

#define C { \
{0, 0, 1, 0, 0}, \
{0, 1, 1, 1, 0}, \
{1, 1, 0, 0, 1}, \
{1, 1, 0, 0, 1}, \
{1, 1, 0, 0, 1}, \
{0, 1, 1, 1, 0}, \
{0, 0, 1, 0, 0} \
}

#define D { \
{0, 0, 1, 0, 0}, \
{0, 1, 1, 1, 0}, \
{0, 1, 0, 0, 1}, \
{0, 1, 0, 0, 1}, \
{0, 1, 0, 0, 1}, \
{0, 1, 1, 1, 0}, \
{0, 0, 1, 0, 0} \
}

#define E { \
{0, 0, 1, 0, 0}, \
{0, 0, 1, 1, 0}, \
{0, 0, 1, 0, 1}, \
{0, 0, 1, 0, 1}, \
{0, 0, 1, 0, 1}, \
{0, 0, 1, 1, 0}, \
{0, 0, 1, 0, 0} \
}

#define F { \
{0, 0, 1, 0, 0}, \
{0, 0, 1, 1, 0}, \
{0, 0, 0, 1, 1}, \
{0, 0, 0, 1, 1}, \
{0, 0, 0, 1, 1}, \
{0, 0, 1, 1, 0}, \
{0, 0, 1, 0, 0} \
}

#define G { \
{0, 0, 1, 0, 0}, \
{0, 0, 0, 1, 0}, \
{0, 0, 0, 0, 1}, \
{0, 0, 0, 0, 1}, \
{0, 0, 0, 0, 1}, \
{0, 0, 0, 1, 0}, \
{0, 0, 1, 0, 0} \
}

#define small_O { \
{0, 0, 1, 0, 0}, \
{0, 0, 1, 0, 0}, \
{0, 0, 1, 1, 1}, \
{0, 0, 1, 1, 1}, \
{0, 0, 1, 1, 1}, \
{0, 0, 1, 0, 0}, \
{0, 0, 1, 0, 0} \
}
```

*CODE*

```
#define small_W { \
{0, 0, 1, 0, 0}, \
{0, 0, 1, 1, 0}, \
{0, 0, 0, 1, 1}, \
{0, 0, 0, 1, 1}, \
{0, 0, 0, 1, 1}, \
{0, 0, 1, 1, 0}, \
{0, 0, 1, 0, 0} \
}

#define small_R { \
{0, 0, 0, 1, 0}, \
{0, 0, 0, 1, 0}, \
{0, 0, 0, 1, 1}, \
{0, 0, 0, 1, 1}, \
{0, 0, 0, 1, 1}, \
{0, 0, 0, 1, 0}, \
{0, 0, 0, 1, 0} \
}

#define small_D { \
{0, 0, 0, 0, 1}, \
{0, 0, 0, 0, 1}, \
{0, 1, 1, 0, 1}, \
{1, 0, 0, 1, 1}, \
{1, 0, 0, 0, 1}, \
{1, 0, 0, 0, 1}, \
{0, 1, 1, 1, 1} \
}

#define COLS 5
#define ROWS 7
#define PINS 13

// For LED Matrix Common Anode choose Current=0;
// For LED Matrix Common Cathode choose Current=1;
int Current=1;

// pin[xx] on led matrix connected to nn on Arduino (-1 is dummy to make array start at pos 1)
int pins[PINS]= {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
// col[xx] of leds = pin yy on led matrix
int cols[COLS] = {pins[1], pins[3], pins[10], pins[7], pins[8]};

// row[xx] of leds = pin yy on led matri
int rows[ROWS] = {pins[12], pins[11], pins[2], pins[9], pins[4], pins[5],
pins[6]};

const int numPatterns = 24;
byte patterns[numPatterns][7][5] = {
  A, B, C, D, E, F,
  G, F, E, D, C, B,
  A, A, A, A, A, A,
  A, A, A, A, A, A
};

int pattern = 0;
boolean normal;

void setup()
{
  //  Switch the current flow in Rows and Columns
    if (Current==1) {
     normal=1;
    } else {
   normal=0;
}
  // sets the pins as output
  for (int i = 1; i <= 12; i++) {
    pinMode(pins[i], OUTPUT);
  }
```

*CODE*

```
  // set up cols and rows
  for (int i = 1; i <= colNum; i++) {
    digitalWrite(cols[i - 1], normal);
  }

  for (int i = 1; i <= rowNum; i++) {
    digitalWrite(rows[i - 1], normal);
  }

  clearLeds();

  // Turn off toggling of pin 11
  FrequencyTimer2::disable();
  // Set refresh rate (interrupt timeout period)
  FrequencyTimer2::setPeriod(2000);
  // Set interrupt routine to be called
  FrequencyTimer2::setOnOverflow(display);

  setPattern(pattern);
}

void loop()
{
  pattern = ++pattern % numPatterns;
  setPattern(pattern);
  delay(50);
}

void clearLeds()
{
  // Clear display array
  for (int i = 0; i < colNum; i++) {
    for (int j = 0; j < rowNum; j++) {
      leds[i][j] = 0;
    }
  }
}

void setPattern(int pattern)
{
  for (int i = 0; i < colNum; i++) {
    for (int j = 0; j < rowNum; j++) {
      leds[i][j] = patterns[pattern][j][i];
    }
  }
}


// Interrupt routine
void display()
{
  // Turn whole previous column off:
  digitalWrite(cols[col], normal);
  col++;
  if (col == colNum) {
    col = 0;
  }

  for (int row = 0; row < rowNum; row++) {
    if (leds[col][row] == 1) {
      digitalWrite(rows[row], normal);
    }
    else {
      digitalWrite(rows[row], !normal); // reverse the normal
    }
  }
  // Turn whole column on at once (for equal lighting times):
  digitalWrite(cols[col], !normal); // reverse the normal
}
```

# Interaction with 8X8 LED Matrix

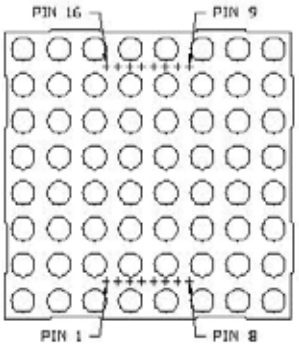
http://arduino.cc/en/Tutorial/LiquidCrystalTextDirection

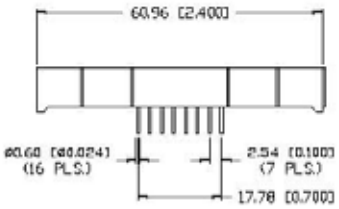
LED displays are often packaged as matrixes of LEDs arranged in rows of common anodes and columns of common cathodes, or the reverse.

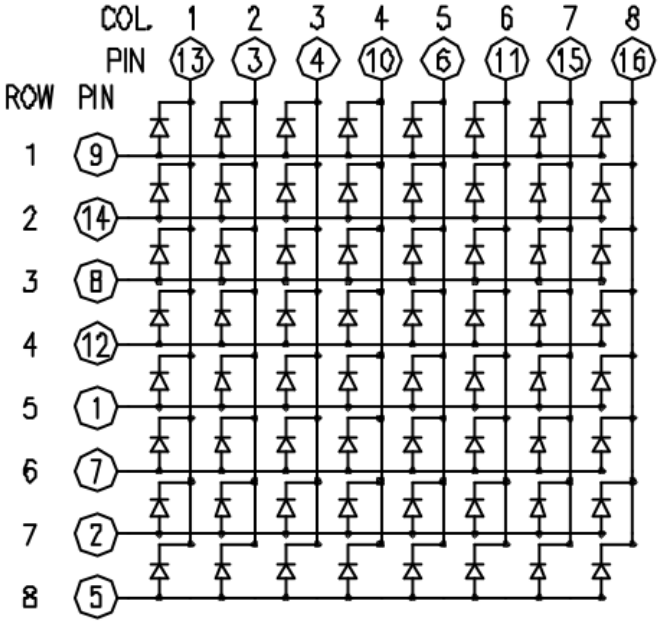
The 16 pins of the matrix are hooked up to 16 pins of the Arduino. Four of the analog pins are used as digital inputs 16 through 19. The order of the pins is assigned in two arrays in the code.
Two potentiometers, connected to analog pins 0 and 1, control the movement of a lit LED in the matrix.

*HARDWARE REQUIRED*

Arduino Board
(1) 8 x 8 LED Matrix
(2) potentiometers
hook-up wire
breadboard



*8X8 LED MATRIX CONNECTION WITH ARDUINO*

These can be very useful displays. To control a matrix, you connect both its rows and columns to your microcontroller. The columns are connected to the LEDs anodes (see Figure 1), so a column needs to be high for any of the LEDs in that column to turn on. The rows are connected to the LEDs cathodes, so the row needs to be low for an individual LED to turn on. If the row and the column are both high or both low, no voltage flows through the LED and it doesn't turn on.

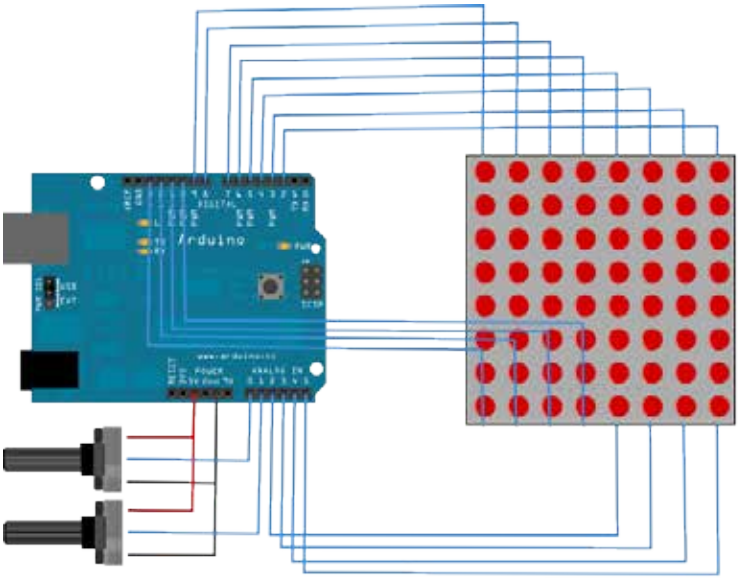
To control an individual LED, you set its column high and its row low. To control multiple LEDs in a row, you set the rows high, then take the column high, then set the lows row or high as appropriate; a low row will turn the corresponding LED on, and a high row will turn it off.

Although there are pre-made LED matrices, you can also make your own matrix from 64 LEDs, using the schematic as shown above.
It doesn't matter which pins of the microcontroller you connect the rows and columns to, because you can assign things in software. Connected the pins in a way that makes wiring easiest. A typical layout is shown below.

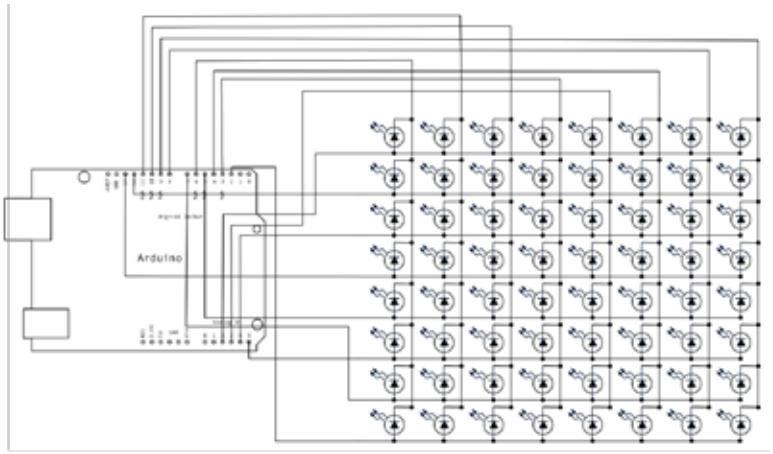
Here's a matrix of the pin connections, based on the diagram above:

| Matrix pin no. | Row | Column | Arduino pin number |
|---|---|---|---|
| 1 | 5 | - | 13 |
| 2 | 7 | - | 12 |
| 3 | - | 2 | 11 |
| 4 | - | 3 | 10 |
| 5 | 8 | - | 16 (analog pin 2) |
| 6 | - | 5 | 17 (analog pin 3) |
| 7 | 6 | - | 18 (analog pin 4) |
| 8 | 3 | - | 19 (analog pin 5) |
| 9 | 1 | - | 2 |
| 10 | - | 4 | 3 |
| 11 | - | 6 | 4 |
| 12 | 4 | - | 5 |
| 13 | - | 1 | 6 |
| 14 | 2 | - | 7 |
| 15 | - | 7 | 8 |
| 16 | - | 8 | 9 |

*IMAGE*



*SCHEMATIC*



*CODE*

```
 #include <FrequencyTimer2.h>

/ 2-dimensional array of row pin numbers:
const int row[8] = {
  2,7,19,5,13,18,12,16 };
// 2-dimensional array of column pin numbers:
const int col[8] = {
  6,11,10,3,17,4,8,9  };

// 2-dimensional array of pixels:
int pixels[8][8];

// cursor position:
int x = 5;
int y = 5;

void setup() {
  // initialize the I/O pins as outputs
  // iterate over the pins:
  for (int thisPin = 0; thisPin < 8; thisPin++) {
    // initialize the output pins:
    pinMode(col[thisPin], OUTPUT);
    pinMode(row[thisPin], OUTPUT);
    // take the col pins (i.e. the cathodes) high to ensure that
    // the LEDS are off:
    digitalWrite(col[thisPin], HIGH);
  }
```

*CODE*

```
/*
 Row-Column Scanning an 8x8 LED matrix with
X-Y input

 This example controls an 8x8 LED matrix using
two analog inputs

 created 27 May 2009
 modified 30 Aug 2011
 by Tom Igoe

 This example works for the Lumex  LDM-
24488NI Matrix. See
 http://sigma.octopart.com/140413/datasheet/
Lumex-LDM-24488NI.pdf
 for the pin connections

 For other LED cathode column matrixes, you
should only need to change
 the pin numbers in the row[] and column[]
arrays

 rows are the anodes
 cols are the cathodes
 ---------

 Pin numbers:
 Matrix:
 * Digital pins 2 through 13,
 * analog pins 2 through 5 used as digital 16
through 19
 Potentiometers:
 * center pins are attached to analog pins 0 and
1, respectively
 * side pins attached to +5V and ground,
respectively.

 This example code is in the public domain.
 http://www.arduino.cc/en/Tutorial/
RowColumnScanning
  see also
http://www.tigoe.net/pcomp/code/
category/arduinowiring/514
for more
 */
```

```
  // initialize the pixel matrix:
  for (int x = 0; x < 8; x++) {
    for (int y = 0; y < 8; y++) {
      pixels[x][y] = HIGH;
    }
  }
}

void loop() {
  // read input:
  readSensors();

  // draw the screen:
  refreshScreen();
}

void readSensors() {
  // turn off the last position:
  pixels[x][y] = HIGH;
  // read the sensors for X and Y values:
  x = 7 - map(analogRead(A0), 0, 1023, 0, 7);
  y = map(analogRead(A1), 0, 1023, 0, 7);
  // set the new pixel position low so that the LED will turn on
  // in the next screen refresh:
  pixels[x][y] = LOW;

}

void refreshScreen() {
  // iterate over the rows (anodes):
  for (int thisRow = 0; thisRow < 8; thisRow++) {
    // take the row pin (anode) high:
    digitalWrite(row[thisRow], HIGH);
    // iterate over the cols (cathodes):
    for (int thisCol = 0; thisCol < 8; thisCol++) {
      // get the state of the current pixel;
      int thisPixel = pixels[thisRow][thisCol];
      // when the row is HIGH and the col is LOW,
      // the LED where they meet turns on:
      digitalWrite(col[thisCol], thisPixel);
      // turn the pixel off:
      if (thisPixel == LOW) {
        digitalWrite(col[thisCol], HIGH);
      }
    }
    // take the row pin low to turn off the whole row:
    digitalWrite(row[thisRow], LOW);
  }
}
```

# Control Scroll Text Speed on 8X8 LED Matrix

http://playground.arduino.cc/Main/DirectDriveLEDMatrix

This example scroll to the left the "Hello" message.

*HARDWARE REQUIRED*

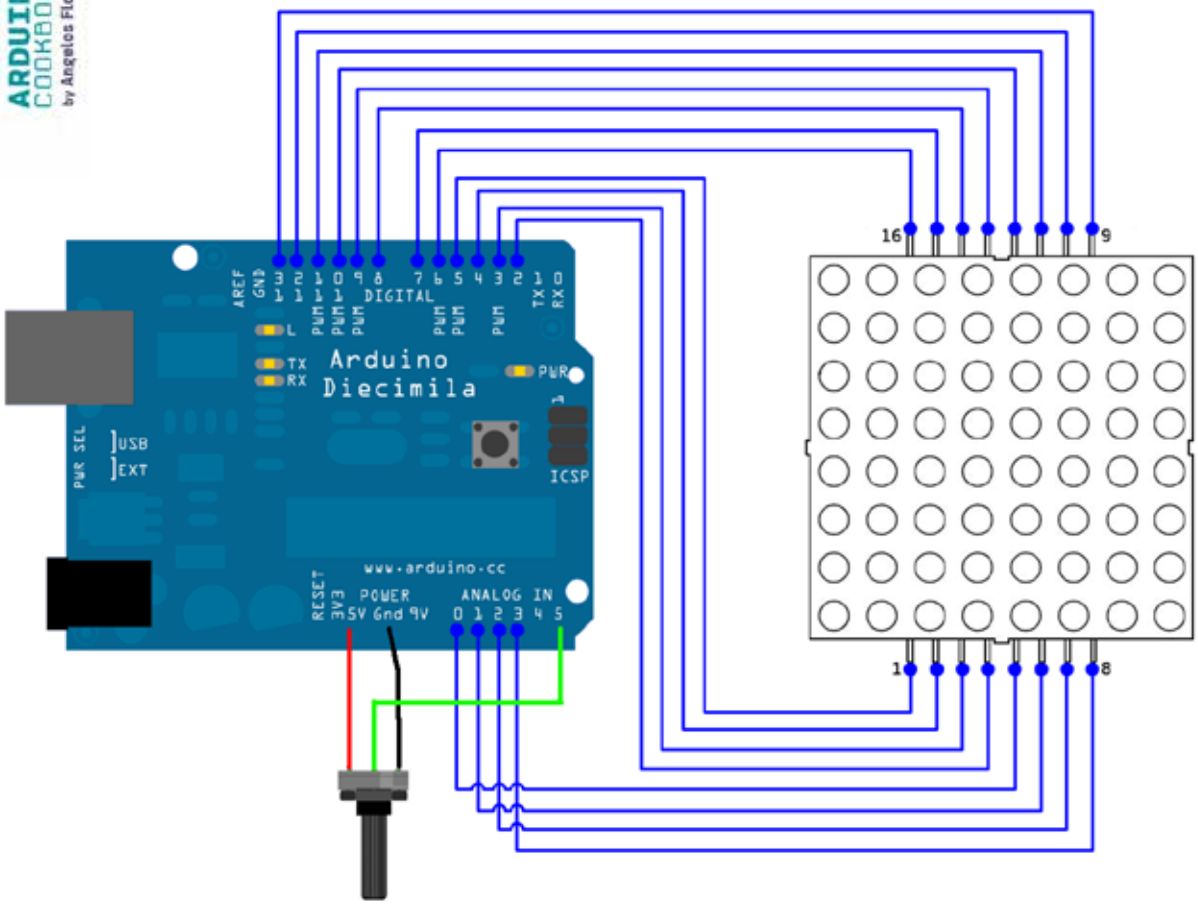
Arduino Board
(1) 8 x 8 LED Matrix
(16) 220 Ohm resistors
hook-up wire
breadboard

*CIRCUIT*

Direct wiring an Arduino to an LED matrix

```
// pin[xx] on led matrix connected to nn on Arduino (-1 is dummy to make array start at pos 1)
int pins[17]= {-1, 5, 4, 3, 2, 14, 15, 16, 17, 13, 12, 11, 10, 9, 8, 7, 6};
```

*IMAGE*



*CODE*

```
#include <FrequencyTimer2.h>

#define SPACE { \
   {0, 0, 0, 0, 0, 0, 0, 0}, \
   {0, 0, 0, 0, 0, 0, 0, 0}, \
   {0, 0, 0, 0, 0, 0, 0, 0}, \
   {0, 0, 0, 0, 0, 0, 0, 0}, \
   {0, 0, 0, 0, 0, 0, 0, 0}, \
   {0, 0, 0, 0, 0, 0, 0, 0}, \
   {0, 0, 0, 0, 0, 0, 0, 0}, \
   {0, 0, 0, 0, 0, 0, 0, 0} \
}

#define H { \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 1, 1, 1, 1, 1, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}  \
}

#define E  { \
   {0, 1, 1, 1, 1, 1, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 1, 1, 1, 1, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 1, 1, 1, 1, 1, 0}  \
}

#define L { \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 0, 0}, \
   {0, 1, 1, 1, 1, 1, 1, 0}  \
}

#define O { \
   {0, 0, 0, 1, 1, 0, 0, 0}, \
   {0, 0, 1, 0, 0, 1, 0, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 1, 0, 0, 0, 0, 1, 0}, \
   {0, 0, 1, 0, 0, 1, 0, 0}, \
   {0, 0, 0, 1, 1, 0, 0, 0}  \
}
```

```
/*
 * Show messages on an 8x8 led
matrix,
 * scrolling from right to left.
 *
 * Uses FrequencyTimer2 library to
 * constantly run an interrupt
routine
 * at a specified frequency. This
 * refreshes the display without
the
 * main loop having to do any-
thing.
 *
 */
```

*CODE*

```
byte col = 0;
byte leds[8][8];

// pin[xx] on led matrix connected to nn on Arduino (-1 is dummy to make array start at pos 1)
int pins[17]= {-1, 5, 4, 3, 2, 14, 15, 16, 17, 13, 12, 11, 10, 9, 8, 7, 6};
// col[xx] of leds = pin yy on led matrix
int cols[8] = {pins[13], pins[3], pins[4], pins[10], pins[06], pins[11], pins[15], pins[16]};

// row[xx] of leds = pin yy on led matrix
int rows[8] = {pins[9], pins[14], pins[8], pins[12], pins[1], pins[7], pins[2], pins[5]};

const int analogInPin = A5;  // Analog input pin that the potentiometer is attached to
const int numPatterns = 6;

int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM (analog out)

byte patterns[numPatterns][8][8] = {H,E,L,L,O,SPACE};

int pattern = 0;

void setup() {
  Serial.begin(9600);
  // sets the pins as output
  for (int i = 1; i <= 16; i++) {
    pinMode(pins[i], OUTPUT);
  }

  // set up cols and rows
  for (int i = 1; i <= 8; i++) {
    digitalWrite(cols[i - 1], HIGH);
  }

  for (int i = 1; i <= 8; i++) {
    digitalWrite(rows[i - 1], HIGH);
  }

  clearLeds();

  // Turn off toggling of pin 11
  FrequencyTimer2::disable();
  // Set refresh rate (interrupt timeout period)
  FrequencyTimer2::setPeriod(2000);
  // Set interrupt routine to be called
  FrequencyTimer2::setOnOverflow(display);

  setPattern(pattern);
}

void loop() {
    // read the analog in value:
  sensorValue = analogRead(analogInPin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 20, 100);


    pattern = ++pattern % numPatterns;
    slidePattern(pattern, outputValue);

  Serial.print("pattern = " );
  Serial.print(pattern);
  Serial.print("\t sensorValue = ");
  Serial.print(sensorValue);
  Serial.print("\t outputValue = ");
  Serial.println(outputValue);
}
```

*CODE*

```
void clearLeds() {
  // Clear display array
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      leds[i][j] = 0;
    }
  }
}

void setPattern(int pattern) {
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      leds[i][j] = patterns[pattern][i][j];
    }
  }
}

void slidePattern(int pattern, int del) {
  for (int l = 0; l < 8; l++) {
    for (int i = 0; i < 7; i++) {
      for (int j = 0; j < 8; j++) {
        leds[j][i] = leds[j][i+1];
      }
    }
    for (int j = 0; j < 8; j++) {
      leds[j][7] = patterns[pattern][j][0 + l];
    }
    delay(del);
  }
}

// Interrupt routine
void display() {


  digitalWrite(cols[col], HIGH);  // Turn whole previous column off
  col++;
  if (col == 8) {
    col = 0;
  }
  for (int row = 0; row < 8; row++) {
    if (leds[col][7 - row] == 1) {
      digitalWrite(rows[row], HIGH);  // Turn on this led
    }
    else {
      digitalWrite(rows[row], LOW); // Turn off this led
    }
  }
  digitalWrite(cols[col], LOW);
  // Turn whole column on at once (for equal lighting times)
}
```

## Matrix Project

*EXERCISE*

Use an LED matrix as an main interface item on your project

*HARDWARE REQUIRED*

Arduino Board
LED matrix

*SCHEMATIC*

# Graphics using Nokia 3310/5110 LCD Display

http://www.adafruit.com/products/338
https://learn.adafruit.com/nokia-5110-3310-monochrome-lcd
http://www.thaieasyelec.com/LCD-Display/Module/Graphic-LCD-84-48-Nokia-5110.html
http://playground.arduino.cc/Code/PCD8544
http://nds2.nokia.com/files/support/apac/phones/guides/Nokia_5110_APAC_UG__EN.pdf

CONNECTION: From LCD to Arduino Pins

| | |
|---|---|
| pin 7 | Serial clock out (SCLK) |
| pin 6 | Serial data out (DIN) |
| pin 5 | Data/Command select (D/C) |
| pin 4 | LCD chip select (CS) |
| pin 3 | LCD reset (RST) |
| LCD_CMD | 13 (LIGHT) Digital Pin 13 else control LCD backlight with potentiometer using PWM pins |
| VCC | 3.3V (DO NOT PLUG IT TO 5V) |
| GRD | GROUND |

*HARDWARE REQUIRED*

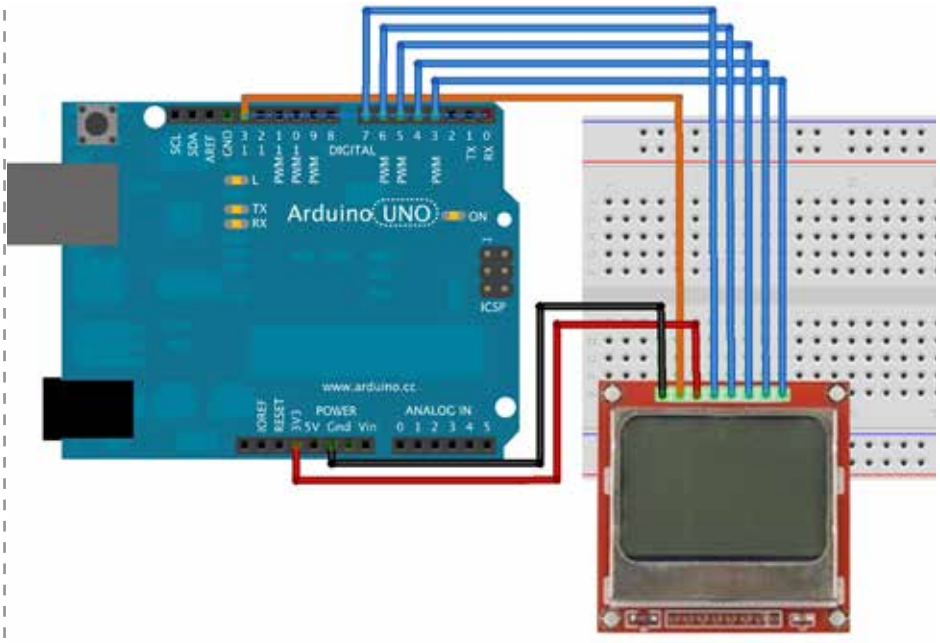
Arduino Board
LCD Display NOKIA type 3310 or 5110

*CIRCUIT*

SOME PROJECTS REQUIRE LIBRARIES TO BE INSTALLED
These displays are small, only about 1.5» diameter, but very readable due and comes with a backlight. This display is made of 84x48 individual pixels, so you can use it for graphics, text or bitmaps. These displays are inexpensive, easy to use, require only a few digital I/O pins and are fairly low power as well.

To drive the display, you will need 3 to 5 digital output pins (depending on whether you want to manually control the chip select and reset lines). Another pin can be used to control (via on/off or PWM) the backlight. To make things easy for you, we've written a nice graphics library that can print text, pixels, rectangles, circles and lines! The library is written for the Arduino but can easily be ported to your favorite microcontroller.

*IMAGE*



*CODE*

```
/*************************************************************
This is an example sketch for our Monochrome Nokia 5110 LCD Displays

  Pick one up today in the adafruit shop!
  ------> http://www.adafruit.com/products/338

These displays use SPI to communicate, 4 or 5 pins are required to interface

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries.
BSD license, check license.txt for more information
All text above, and the splash screen must be included in any redistribution
*************************************************************/

// libraries
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>

// Software SPI (slower updates, more flexible pin options):
// pin 7 - Serial clock out (SCLK)
// pin 6 - Serial data out (DIN)
// pin 5 - Data/Command select (D/C)
// pin 4 - LCD chip select (CS)
// pin 3 - LCD reset (RST)

Adafruit_PCD8544 display = Adafruit_PCD8544(7, 6, 5, 4, 3);

// Hardware SPI (faster, but must use certain hardware pins):
// SCK is LCD serial clock (SCLK) - this is pin 13 on Arduino Uno
// MOSI is LCD DIN - this is pin 11 on an Arduino Uno
// pin 5 - Data/Command select (D/C)
// pin 4 - LCD chip select (CS)
// pin 3 - LCD reset (RST)
// Adafruit_PCD8544 display = Adafruit_PCD8544(5, 4, 3);
// Note with hardware SPI MISO and SS pins aren't used but will still be read
// and written to during SPI transfer.  Be careful sharing these pins!

#define NUMFLAKES 10
#define XPOS 0
#define YPOS 1
#define DELTAY 2

#define LOGO16_GLCD_HEIGHT 16
#define LOGO16_GLCD_WIDTH  16

static unsigned char PROGMEM logo16_glcd_bmp[] =
{ B00000000, B11000000,
  B00000001, B11000000,
  B00000001, B11000000,
  B00000011, B11100000,
  B11110011, B11100000,
  B11111110, B11111000,
  B01111110, B11111111,
  B00110011, B10011111,
  B00011111, B11111100,
  B00001101, B01110000,
  B00011011, B10100000,
  B00111111, B11100000,
  B00111111, B11110000,
  B01111100, B11110000,
  B01110000, B01110000,
  B00000000, B00110000 };

void setup()   {
  Serial.begin(9600);
  pinMode(13,OUTPUT); // backlight connection

display.begin();
  // init done

  // you can change the contrast around to adapt the display
  // for the best viewing!
  display.setContrast(50);

  display.display(); // show splashscreen
  delay(2000);
  display.clearDisplay();   // clears the screen and buffer

  // draw a single pixel
  display.drawPixel(10, 10, BLACK);
  display.display();
  delay(2000);
  display.clearDisplay();
```

*CODE*

```
// draw many lines
testdrawline();
display.display();
delay(2000);
display.clearDisplay();

// draw rectangles
testdrawrect();
display.display();
delay(2000);
display.clearDisplay();

// draw multiple rectangles
testfillrect();
display.display();
delay(2000);
display.clearDisplay();

// draw mulitple circles
testdrawcircle();
display.display();
delay(2000);
display.clearDisplay();

// draw a circle on the center, 10 pixel radius
display.fillCircle(display.width()/2, display.height()/2, 10, BLACK);
display.display();
delay(2000);
display.clearDisplay();

testdrawroundrect();
delay(2000);
display.clearDisplay();

testfillroundrect();
delay(2000);
display.clearDisplay();

testdrawtriangle();
delay(2000);
display.clearDisplay();

testfilltriangle();
delay(2000);
display.clearDisplay();

// draw the first ~12 characters in the font
testdrawchar();
display.display();
delay(2000);
display.clearDisplay();

// text display tests
display.setTextSize(1);
display.setTextColor(BLACK);
display.setCursor(0,0);
display.println("Hello, world!");
display.setTextColor(WHITE, BLACK); // 'inverted' text
display.println(3.141592);
display.setTextSize(2);
display.setTextColor(BLACK);
display.print("0x"); display.println(0xDEADBEEF, HEX);
display.display();
delay(2000);

// miniature bitmap display
display.clearDisplay();
display.drawBitmap(30, 16,  logo16_glcd_bmp, 16, 16, 1);
display.display();
```

*CODE*

```
  // invert the display
  display.invertDisplay(true);
  delay(1000);
  display.invertDisplay(false);
  delay(1000);

  // draw a bitmap icon and 'animate' movement
  testdrawbitmap(logo16_glcd_bmp, LOGO16_GLCD_HEIGHT, LOGO16_
GLCD_WIDTH);
}


void loop() {
  digitalWrite(13,HIGH);
}


void testdrawbitmap(const uint8_t *bitmap, uint8_t w, uint8_t h) {
  uint8_t icons[NUMFLAKES][3];
  srandom(666);     // whatever seed

  // initialize
  for (uint8_t f=0; f< NUMFLAKES; f++) {
    icons[f][XPOS] = random() % display.width();
    icons[f][YPOS] = 0;
    icons[f][DELTAY] = random() % 5 + 1;

    Serial.print("x: ");
    Serial.print(icons[f][XPOS], DEC);
    Serial.print(" y: ");
    Serial.print(icons[f][YPOS], DEC);
    Serial.print(" dy: ");
    Serial.println(icons[f][DELTAY], DEC);
  }

  while (1) {
    // draw each icon
    for (uint8_t f=0; f< NUMFLAKES; f++) {
      display.drawBitmap(icons[f][XPOS], icons[f][YPOS], logo16_glcd_
bmp, w, h, BLACK);
    }
    display.display();
    delay(200);

    // then erase it + move it
    for (uint8_t f=0; f< NUMFLAKES; f++) {
      display.drawBitmap(icons[f][XPOS], icons[f][YPOS],  logo16_glcd_
bmp, w, h, WHITE);
      // move it
      icons[f][YPOS] += icons[f][DELTAY];
      // if its gone, reinit
      if (icons[f][YPOS] > display.height()) {
          icons[f][XPOS] = random() % display.width();
          icons[f][YPOS] = 0;
          icons[f][DELTAY] = random() % 5 + 1;
      }
    }
   }
}


void testdrawchar(void) {
  display.setTextSize(1);
  display.setTextColor(BLACK);
  display.setCursor(0,0);

  for (uint8_t i=0; i < 168; i++) {
    if (i == '\n') continue;
    display.write(i);
```

```
    //if ((i > 0) && (i % 14 == 0))
      //display.println();
  }
  display.display();
}

void testdrawcircle(void) {
  for (int16_t i=0; i<display.height(); i+=2) {
    display.drawCircle(display.width()/2, display.height()/2, i, BLACK);
    display.display();
  }
}

void testfillrect(void) {
  uint8_t color = 1;
  for (int16_t i=0; i<display.height()/2; i+=3) {
    // alternate colors
    display.fillRect(i, i, display.width()-i*2, display.height()-i*2, color%2);
    display.display();
    color++;
  }
}

void testdrawtriangle(void) {
  for (int16_t i=0; i<min(display.width(),display.height())/2; i+=5) {
    display.drawTriangle(display.width()/2, display.height()/2-i,
                  display.width()/2-i, display.height()/2+i,
                  display.width()/2+i, display.height()/2+i, BLACK);
    display.display();
  }
}

void testfilltriangle(void) {
  uint8_t color = BLACK;
  for (int16_t i=min(display.width(),display.height())/2; i>0; i-=5) {
    display.fillTriangle(display.width()/2, display.height()/2-i,
                  display.width()/2-i, display.height()/2+i,
                  display.width()/2+i, display.height()/2+i, color);
    if (color == WHITE) color = BLACK;
    else color = WHITE;
    display.display();
  }
}

void testdrawroundrect(void) {
  for (int16_t i=0; i<display.height()/2-2; i+=2) {
    display.drawRoundRect(i, i, display.width()-2*i, display.height()-2*i,
display.height()/4, BLACK);
    display.display();
  }
}

void testfillroundrect(void) {
  uint8_t color = BLACK;
  for (int16_t i=0; i<display.height()/2-2; i+=2) {
    display.fillRoundRect(i, i, display.width()-2*i, display.height()-2*i,
display.height()/4, color);
    if (color == WHITE) color = BLACK;
    else color = WHITE;
    display.display();
  }
}

void testdrawrect(void) {
  for (int16_t i=0; i<display.height()/2; i+=2) {
    display.drawRect(i, i, display.width()-2*i, display.height()-2*i, BLACK);
    display.display();
  }
}
```

*CODE*

```
void testdrawline() {
  for (int16_t i=0; i<display.width(); i+=4) {
    display.drawLine(0, 0, i, display.height()-1, BLACK);
    display.display();
  }
  for (int16_t i=0; i<display.height(); i+=4) {
    display.drawLine(0, 0, display.width()-1, i, BLACK);
    display.display();
  }
  delay(250);

  display.clearDisplay();
  for (int16_t i=0; i<display.width(); i+=4) {
    display.drawLine(0, display.height()-1, i, 0, BLACK);
    display.display();
  }
  for (int8_t i=display.height()-1; i>=0; i-=4) {
    display.drawLine(0, display.height()-1, display.width()-1, i, BLACK);
    display.display();
  }
  delay(250);

  display.clearDisplay();
  for (int16_t i=display.width()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, i, 0, BLACK);
    display.display();
  }
  for (int16_t i=display.height()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, 0, i, BLACK);
    display.display();
  }
  delay(250);

  display.clearDisplay();
  for (int16_t i=0; i<display.height(); i+=4) {
    display.drawLine(display.width()-1, 0, 0, i, BLACK);
    display.display();
  }
  for (int16_t i=0; i<display.width(); i+=4) {
    display.drawLine(display.width()-1, 0, i, display.height()-1, BLACK);
    display.display();
  }
  delay(250);
}
```

# Timer using  Nokia 3310/5110 LCD Display

https://github.com/moozzyk/ArduinoDigitalClockhttp://www.thaieasyelec.com/
http://blog.3d-logic.com/
http://www.youtube.com/watch?v=-60g9EO3W8o

CONNECTION: From LCD to Arduino Pins

| | |
|---|---|
| pin 7 | Serial clock out (SCLK) |
| pin 6 | Serial data out (DIN) |
| pin 5 | Data/Command select (D/C) |
| pin 4 | LCD chip select (CS) or (CE) |
| pin 3 | LCD reset (RST) |
| LCD_CMD | 13 (LIGHT) Digital Pin 13 else control LCD backlight with potentiometer using PWM pins |
| VCC | 3.3V (DO NOT PLUG IT TO 5V) |
| GRD | GROUND |

*HARDWARE REQUIRED*

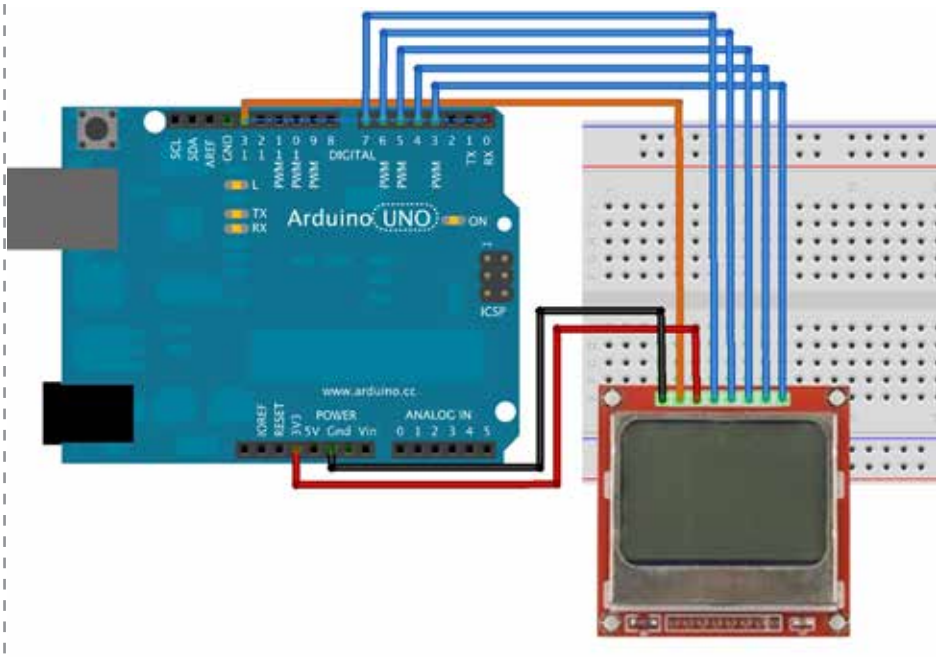
Arduino Board
LCD Display NOKIA type 3310 or 5110

*CIRCUIT*

SOME PROJECTS REQUIRE LIBRARIES TO BE INSTALLED
These displays are small, only about 1.5» diameter, but very readable due and comes with a backlight. This display is made of 84x48 individual pixels, so you can use it for graphics, text or bitmaps. These displays are inexpensive, easy to use, require only a few digital I/O pins and are fairly low power as well.

To drive the display, you will need 3 to 5 digital output pins (depending on whether you want to manually control the chip select and reset lines). Another pin can be used to control (via on/off or PWM) the backlight. To make things easy for you, we've written a nice graphics library that can print text, pixels, rectangles, circles and lines! The library is written for the Arduino but can easily be ported to your favorite microcontroller.

*IMAGE*



*CODE*

```
#define PIN_SCLK  7
#define PIN_SDIN  6
#define PIN_DC    5
#define PIN_SCE   4 // or CE
#define PIN_RESET 3
#define LCD_CMD   0

#define LCD_C     LOW
#define LCD_D     HIGH

#define LCD_X     84
#define LCD_Y     48

static const byte Digits[][4][18] =
{
 {
   { 0xE0, 0xF0, 0xF8, 0xF4, 0xEE, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F,
0x1F, 0x1F, 0xEE, 0xF4, 0xF8, 0xF0, 0xE0 },
   { 0x1F, 0x3F, 0x7F, 0x3F, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x1F, 0x3F, 0x7F, 0x3F, 0x1F },
   { 0xFC, 0xFE, 0xFF, 0xFE, 0xFC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xFC, 0xFE, 0xFF, 0xFE, 0xFC },
   { 0x03, 0x07, 0x0F, 0x17, 0x3B, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C,
0x7C, 0x7C, 0x7C, 0x3B, 0x17, 0x0F, 0x07, 0x03 },
  },
  {
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xE0, 0xF0, 0xF8, 0xF0, 0xE0 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x1F, 0x3F, 0x7F, 0x3F, 0x1F },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xFC, 0xFE, 0xFF, 0xFE, 0xFC },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x03, 0x07, 0x0F, 0x07, 0x03 },
  },
  {
   { 0x00, 0x00, 0x00, 0x04, 0x0E, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F,
0x1F, 0x1F, 0xEE, 0xF4, 0xF8, 0xF0, 0xE0 },
   { 0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0,
0xE0, 0xE0, 0xDF, 0xBF, 0x7F, 0x3F, 0x1F },
   { 0xFC, 0xFE, 0xFF, 0xFE, 0xFD, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00 },
   { 0x03, 0x07, 0x0F, 0x17, 0x3B, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C,
0x7C, 0x7C, 0x7C, 0x38, 0x10, 0x00, 0x00, 0x00 },
  },
  {
   { 0x00, 0x00, 0x00, 0x04, 0x0E, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F,
0x1F, 0x1F, 0xEE, 0xF4, 0xF8, 0xF0, 0xE0 },
   { 0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0,
0xE0, 0xE0, 0xDF, 0xBF, 0x7F, 0x3F, 0x1F },
   { 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0xFD, 0xFE, 0xFF, 0xFE, 0xFC },
   { 0x00, 0x00, 0x00, 0x10, 0x38, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C,
0x7C, 0x7C, 0x7C, 0x3B, 0x17, 0x0F, 0x07, 0x03 },
  },
  {
   { 0xE0, 0xF0, 0xF8, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xE0, 0xF0, 0xF8, 0xF0, 0xE0 },
   { 0x1F, 0x3F, 0x7F, 0xBF, 0xDF, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0,
0xE0, 0xE0, 0xDF, 0xBF, 0x7F, 0x3F, 0x1F },
   { 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0xFD, 0xFE, 0xFF, 0xFE, 0xFC },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x03, 0x07, 0x0F, 0x07, 0x03 },
  },
  {
   { 0xE0, 0xF0, 0xF8, 0xF4, 0xEE, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F,
0x1F, 0x1F, 0x0E, 0x04, 0x00, 0x00, 0x00 },
   { 0x1F, 0x3F, 0x7F, 0xBF, 0xDF, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0,
0xE0, 0xE0, 0xC0, 0x80, 0x00, 0x00, 0x00 },
```

```
/***************************
*****************************
**************
This is an example sketch for our
Monochrome Nokia 5110 LCD
Displays

  Pick one up today in the adafruit
shop!
  ------> http://www.adafruit.com/
products/338

These displays use SPI to commu-
nicate, 4 or 5 pins are required to
interface

Adafruit invests time and re-
sources providing this open source
code,
please support Adafruit and open-
source hardware by purchasing
products from Adafruit!

Written by Limor Fried/Ladyada
for Adafruit Industries.
BSD license, check license.txt for
more information
All text above, and the splash
screen must be included in any
redistribution
*****************************
*****************************
*************/
```

```
    { 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x03, 0xFD, 0xFE, 0xFF, 0xFE, 0xFC },
    { 0x00, 0x00, 0x00, 0x10, 0x38, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C,
0x7C, 0x7C, 0x3B, 0x17, 0x0F, 0x07, 0x03 },
  },
  {
    { 0xE0, 0xF0, 0xF8, 0xF4, 0xEE, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F,
0x1F, 0x0E, 0x04, 0x00, 0x00, 0x00 },
    { 0x1F, 0x3F, 0x7F, 0xBF, 0xDF, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0,
0xE0, 0xE0, 0xC0, 0x80, 0x00, 0x00, 0x00 },
    { 0xFC, 0xFE, 0xFF, 0xFE, 0xFD, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0xFD, 0xFE, 0xFF, 0xFE, 0xFC },
    { 0x03, 0x07, 0x0F, 0x17, 0x3B, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C,
0x7C, 0x7C, 0x3B, 0x17, 0x0F, 0x07, 0x03 },
  },
  {
    { 0x00, 0x00, 0x00, 0x04, 0x0E, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F,
0x1F, 0xEE, 0xF4, 0xF8, 0xF0, 0xE0 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x1F, 0x3F, 0x7F, 0x3F, 0x1F },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xFC, 0xFE, 0xFF, 0xFE, 0xFC },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x03, 0x07, 0x0F, 0x07, 0x03 },
  },
  {
    { 0xE0, 0xF0, 0xF8, 0xF4, 0xEE, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F,
0x1F, 0xEE, 0xF4, 0xF8, 0xF0, 0xE0 },
    { 0x1F, 0x3F, 0x7F, 0xBF, 0xDF, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0,
0xE0, 0xE0, 0xDF, 0xBF, 0x7F, 0x3F, 0x1F },
    { 0xFC, 0xFE, 0xFF, 0xFE, 0xFD, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0xFD, 0xFE, 0xFF, 0xFE, 0xFC },
    { 0x03, 0x07, 0x0F, 0x17, 0x3B, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C,
0x7C, 0x7C, 0x3B, 0x17, 0x0F, 0x07, 0x03 },
  },
  {
    { 0xE0, 0xF0, 0xF8, 0xF4, 0xEE, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F,
0x1F, 0xEE, 0xF4, 0xF8, 0xF0, 0xE0 },
    { 0x1F, 0x3F, 0x7F, 0xBF, 0xDF, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0,
0xE0, 0xE0, 0xDF, 0xBF, 0x7F, 0x3F, 0x1F },
    { 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0xFD, 0xFE, 0xFF, 0xFE, 0xFC },
    { 0x00, 0x00, 0x00, 0x10, 0x38, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C,
0x7C, 0x7C, 0x3B, 0x17, 0x0F, 0x07, 0x03 },
  }
};

static const byte SecondIndicator[4] =
{
  0x00, 0x07, 0x70, 0x00
};

void LcdInitialise(void)
{
  pinMode(PIN_SCE, OUTPUT);
  pinMode(PIN_RESET, OUTPUT);
  pinMode(PIN_DC, OUTPUT);
  pinMode(PIN_SDIN, OUTPUT);
  pinMode(PIN_SCLK, OUTPUT);
  digitalWrite(PIN_RESET, LOW);
  digitalWrite(PIN_RESET, HIGH);

  LcdWrite( LCD_CMD, 0x21 ); // LCD Extended Commands.
  LcdWrite( LCD_CMD, 0xC8 ); // Set LCD Vop (Contrast)
  LcdWrite( LCD_CMD, 0x06 ); // Set Temp coefficent
  LcdWrite( LCD_CMD, 0x14 ); // LCD bias mode 1:48

  LcdWrite( LCD_CMD, 0x20 ); // LCD Standard Commands.
  LcdWrite( LCD_CMD, 0x0C ); // LCD in normal mode. 0x0d for inverse
}
```

```
void LcdWrite(byte dc, byte data)
{
  digitalWrite(PIN_DC, dc);
  digitalWrite(PIN_SCE, LOW);
  shiftOut(PIN_SDIN, PIN_SCLK, MSBFIRST, data);
  digitalWrite(PIN_SCE, HIGH);
}

void LcdClear(void)
{
  for (int index = 0; index < LCD_X * LCD_Y / 8; index++)
  {
    LcdWrite(LCD_D, 0x00);
  }
}

void Spacer()
{
  LcdWrite(LCD_D, 0x00);
  LcdWrite(LCD_D, 0x00);
}

void DisplayTime(byte hour, byte minutes, byte seconds)
{
  byte components[4] =
  {
    (byte)(hour / 10),
    (byte)(hour % 10),
    (byte)(minutes / 10),
    (byte)(minutes % 10)
  };

  for(byte row = 0; row < 4; row++)
  {
    LcdWrite(LCD_C, 0x80 | 0);
    LcdWrite(LCD_C, 0x40 | row);

    for(byte digit = 0; digit < 4; digit++)
    {
      for(byte col = 0; col < 18; col++)
      {
        LcdWrite(LCD_D, Digits[components[digit]][row][col]);
      }

      Spacer();

      if(digit == 1) // Display second indicator after the second digit
      {
        DisplaySecondIndicator(row, seconds & 0x01);
      }
    }
  }

  DrawSecondsBar(seconds);
}

void DisplaySecondIndicator(byte row, boolean show)
{
  for(int secondIndicatorSegment = 0; secondIndicatorSegment < 3; secondIndicator-
Segment++)
  {
    if(show)
    {
      LcdWrite(LCD_D, SecondIndicator[row]);
    }
    else     {
      LcdWrite(LCD_D, 0x00); // clear
    }
  }
```

```
  Spacer();
}

void DrawSecondsBar(byte seconds)
{
  // Position the pointer
  LcdWrite(LCD_C, 0x80 | 0x0b);
  LcdWrite(LCD_C, 0x44);

  // Draw the left side of the progress bar box
  LcdWrite(LCD_D, 0xF0);

  for(byte i = 0; i < 59; i++)
  {
    if(i < seconds)
    {
      LcdWrite(LCD_D, 0xF0);
    }
    else
    {
      LcdWrite(LCD_D, 0x90);
    }
  }

  // Draw the right side of the progress bar box
  LcdWrite(LCD_D, 0xF0);
}

byte tcnt2;
unsigned long time = 0; // 86390000;

void setup(void)
{
  SetupInterrupt();
  InitializeDisplay();
}

// Credits for the interrupt setup routine:
// http://popdevelop.com/2010/04/mastering-timer-interrupts-on-the-arduino/
void SetupInterrupt()
{
  /* First disable the timer overflow interrupt while we're configuring */
  TIMSK2 &= ~(1<<TOIE2);

  /* Configure timer2 in normal mode (pure counting, no PWM etc.) */
  TCCR2A &= ~((1<<WGM21) | (1<<WGM20));
  TCCR2B &= ~(1<<WGM22);

  /* Select clock source: internal I/O clock */
  ASSR &= ~(1<<AS2);

  /* Disable Compare Match A interrupt enable (only want overflow) */
  TIMSK2 &= ~(1<<OCIE2A);

  /* Now configure the prescaler to CPU clock divided by 128 */
  TCCR2B |= (1<<CS22)  | (1<<CS20); // Set bits
  TCCR2B &= ~(1<<CS21);             // Clear bit

  /* We need to calculate a proper value to load the timer counter.
   * The following loads the value 131 into the Timer 2 counter register
   * The math behind this is:
   * (CPU frequency) / (prescaler value) = 125000 Hz = 8us.
   * (desired period) / 8us = 125.
   * MAX(uint8) + 1 - 125 = 131;
   */


  /* Save value globally for later reload in ISR */
  tcnt2 = 131;
```

```
  /* Finally load end enable the timer */
  TCNT2 = tcnt2;
  TIMSK2 |= (1<<TOIE2);
}

void InitializeDisplay()
{
  LcdInitialise();
  LcdClear();
}

/*
 * Install the Interrupt Service Routine (ISR) for Timer2 overflow.
 * This is normally done by writing the address of the ISR in the
 * interrupt vector table but conveniently done by using ISR()  */

ISR(TIMER2_OVF_vect) {
  /* Reload the timer */
  TCNT2 = tcnt2;

  time++;
  time = time % 86400000;
}

void loop(void)
{
  unsigned long t = (unsigned long)(time/1000);

  DisplayTime((byte)(t / 3600), (byte)((t / 60) % 60), (byte)(t % 60));
}
```

# Print Text on LCD Character Display

http://arduino.cc/en/Tutorial/LiquidCrystal

The LiquidCrystal library allows you to control LCD displays that are compatible with the Hitachi HD44780 driver. There are many of them out there, and you can usually tell them by the 16-pin interface. This example sketch prints "Hello World!" to the LCD and shows the time in seconds since the Arduino was reset. The pot controls the brightness of the display.

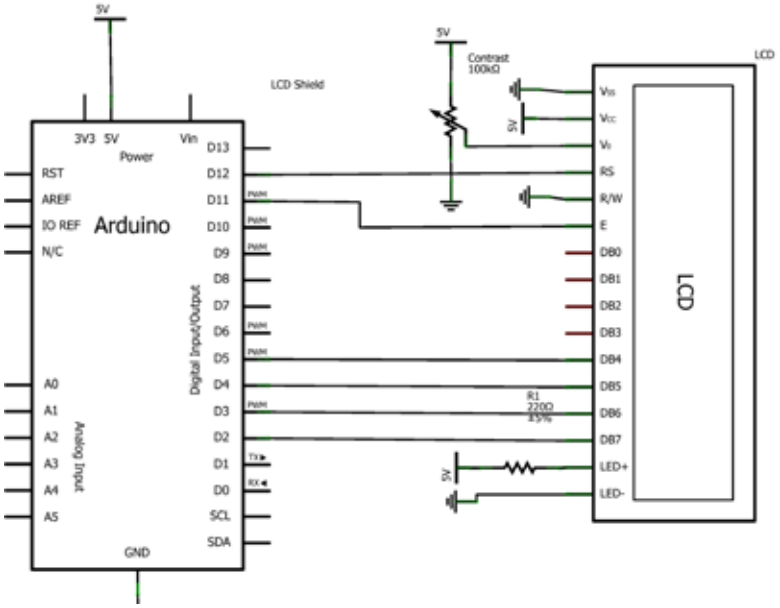
## *HARDWARE REQUIRED*

Arduino Board
LCD Screen (compatible with Hitachi HD44780 driver)
pin headers to solder to the LCD display pins
10k Potentiometer
breadboard
hook-up wire

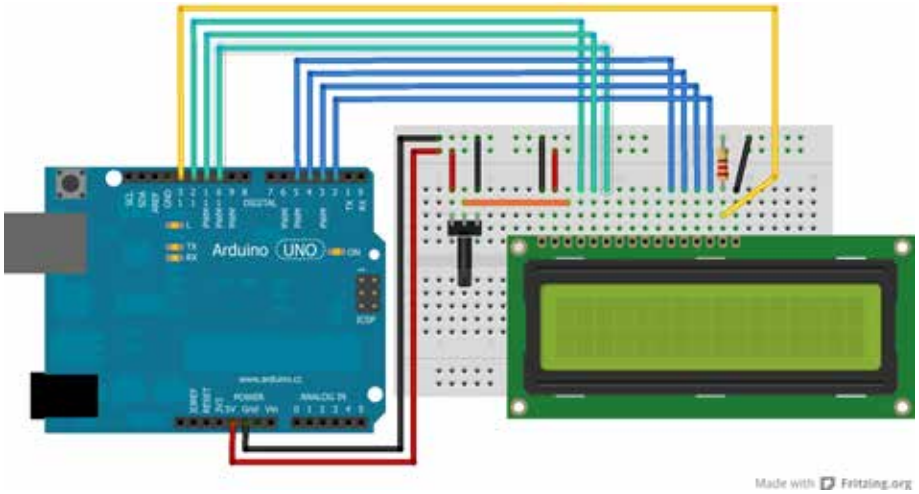
## *CIRCUIT*

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:
A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.
A Read/Write (R/W) pin that selects reading mode or writing mode
An Enable pin that enables writing to the registers
8 data pins (D0 -D7). The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.
There's also a display constrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.
The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The LiquidCrystal Library simplifies this for you so you don't need to know the low-level instructions.
The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

## *SCHEMATIC*



*IMAGE*




*CODE*

```
// character LCD example code
// www.hacktronics.com

#include <LiquidCrystal.h>

// Connections:
// rs (LCD pin 4) to Arduino pin 12
// rw (LCD pin 5) to Arduino pin 11
// enable (LCD pin 6) to Arduino pin 10
// LCD pin 15 to Arduino pin 13
// LCD pins d4, d5, d6, d7 to Arduino pins 5, 4, 3, 2
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);

int backLight = 13;    // pin 13 will control the backlight

void setup()
{
  pinMode(backLight, OUTPUT);
  digitalWrite(backLight, HIGH); // turn backlight on. Replace 'HIGH' with
'LOW' to turn it off.
  lcd.begin(16,2);              // columns, rows.  use 16,2 for a 16x2 LCD, etc.
  lcd.clear();                  // start with a blank screen
  lcd.setCursor(0,0);           // set cursor to column 0, row 0 (the first row)
  lcd.print("Hello, World");  // change this text to whatever you like. keep it
clean.
  lcd.setCursor(0,1);           // set cursor to column 0, row 1
  lcd.print("hacktronics.com");

  // if you have a 4 row LCD, uncomment these lines to write to the bottom rows
  // and change the lcd.begin() statement above.
  //lcd.setCursor(0,2);         // set cursor to column 0, row 2
  //lcd.print("Row 3");
  //lcd.setCursor(0,3);         // set cursor to column 0, row 3
  //lcd.print("Row 4");
}

void loop()
{
}
```

Before wiring the LCD screen to your Arduino we suggest to solder a pin header strip to the 14 (or 16) pin count connector of the LCD screen, as you can see in the image above.

To wire your LCD screen to your Arduino, connect the following pins:
LCD RS pin to digital pin 12
LCD Enable pin to digital pin 11
LCD D4 pin to digital pin 5
LCD D5 pin to digital pin 4
LCD D6 pin to digital pin 3
LCD D7 pin to digital pin 2

Additionally, wire a 10K pot to +5V and GND, with it's wiper (output) to LCD screens VO pin (pin3).

# Serial Print using an LCD Display

http://arduino.cc/en/Tutorial/LiquidCrystalSerial

The Liquid Crystal Library allows you to control LCD displays that are compatible with the Hitachi HD44780 driver. There are many of them out there, and you can usually tell them by the 16-pin interface. This example sketch accepts serial input from a host computer and displays it on the LCD. To use it, upload the sketch, then open the Serial Monitor and type some characters and click Send. The text will appear on your LCD.

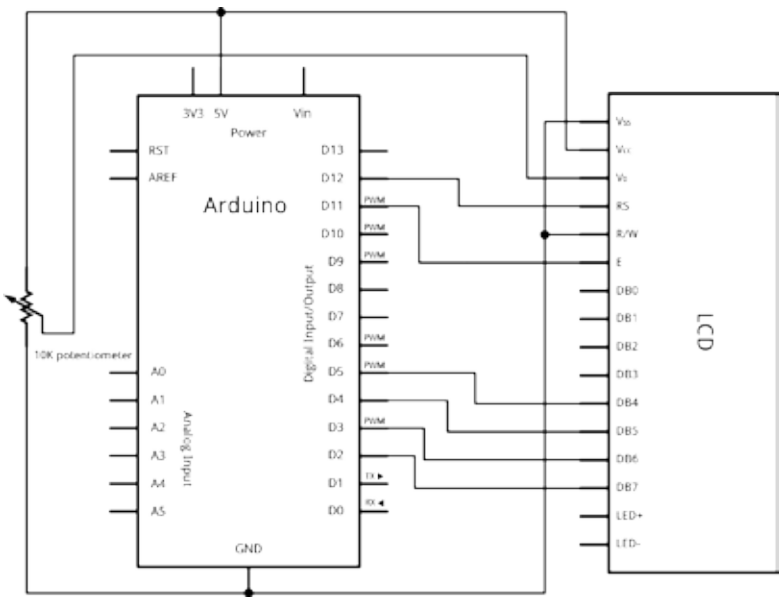
## *HARDWARE REQUIRED*

Arduino Board
LCD Screen (compatible with Hitachi HD44780 driver)
pin headers to solder to the LCD display pins
10k Potentiometer
breadboard
hook-up wire

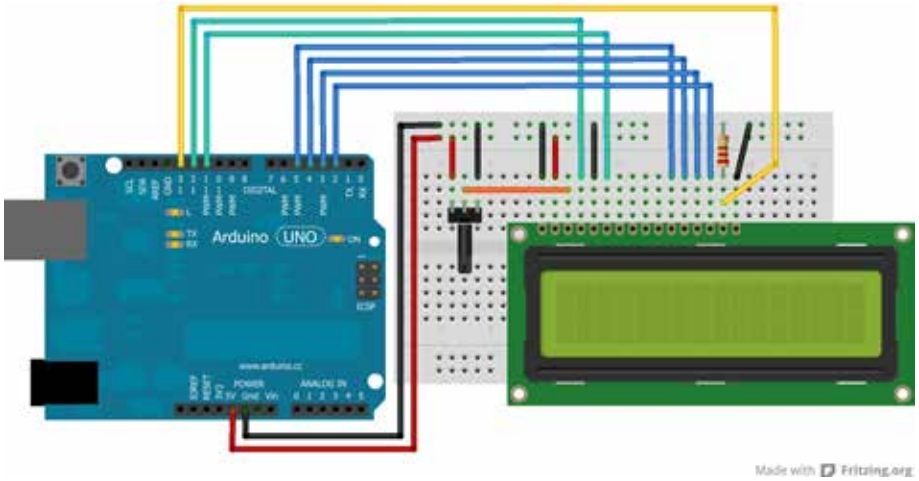
## *CIRCUIT*

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:
A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.
A Read/Write (R/W) pin that selects reading mode or writing mode
An Enable pin that enables writing to the registers
8 data pins (D0 -D7). The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.
There's also a display constrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.
The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The LiquidCrystal Library simplifies this for you so you don't need to know the low-level instructions.
The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

## *SCHEMATIC*



*IMAGE*




*CODE*

```
/*
  LiquidCrystal Library - Serial Input

 Demonstrates the use a 16x2 LCD display.  The LiquidCrystal
 library works with all LCD displays that are compatible with the
 Hitachi HD44780 driver. There are many of them out there, and you
 can usually tell them by the 16-pin interface.

 This sketch displays text sent over the serial port
 (e.g. from the Serial Monitor) on an attached LCD.

 The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)

 Library originally added 18 Apr 2008
 by David A. Mellis
 library modified 5 Jul 2009
 by Limor Fried (http://www.ladyada.net)
 example added 9 Jul 2009
 by Tom Igoe
 modified 22 Nov 2010
 by Tom Igoe

 This example code is in the public domain.

 http://arduino.cc/en/Tutorial/LiquidCrystalSerial
 */

// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup(){
    // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // initialize the serial communications:
  Serial.begin(9600);
}

void loop()
{
  // when characters arrive over the serial port...
  if (Serial.available()) {
    // wait a bit for the entire message to arrive
    delay(100);
    // clear the screen
    lcd.clear();
    // read all the available characters
    while (Serial.available() > 0) {
      // display each character to the LCD
      lcd.write(Serial.read());
    }
  }
}
```

# Scroll Text on LCD Display

http://arduino.cc/en/Tutorial/LiquidCrystalTextDirection

The Liquid Crystal Library allows you to control LCD displays that are compatible with the Hitachi HD44780 driver. There are many of them out there, and you can usually tell them by the 16-pin interface.
This example sketch shows how to use the leftToRight() and right-ToLeft() methods. These methods control which way text flows from the cursor.
rightToLeft() causes text to flow to the left from the cursor, as if the display is right-justified.
leftToRight() causes text to flow to the right from the cursor, as if the display is left-justified.

This sketch prints a through l right to left, then m through r left to right, then s through z right to left again.

## *HARDWARE REQUIRED*

Arduino Board
LCD Screen (compatible with Hitachi HD44780 driver)
pin headers to solder to the LCD display pins
10k Potentiometer
breadboard
hook-up wire

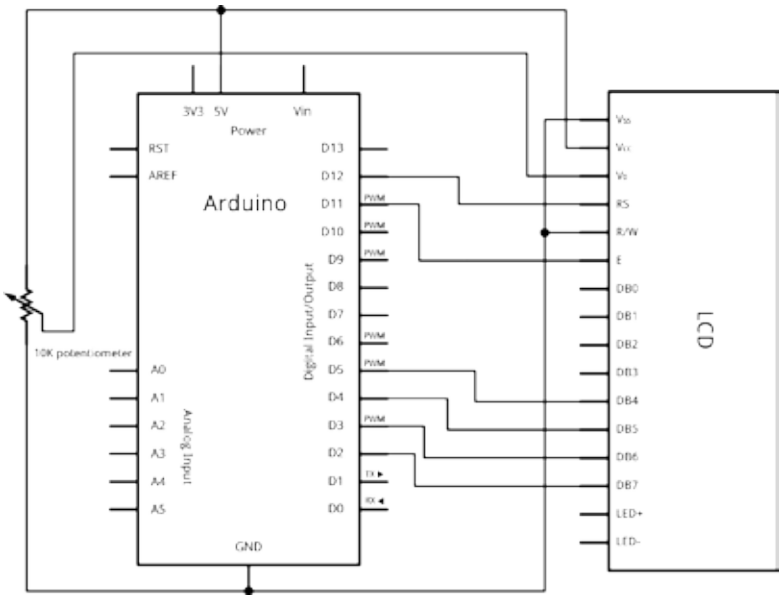
## *CIRCUIT*

Before wiring the LCD screen to your Arduino we suggest to solder a pin header strip to the 14 (or 16) pin count connector of the LCD screen, as you can see in the image above.
To wire your LCD screen to your Arduino, connect the following pins:

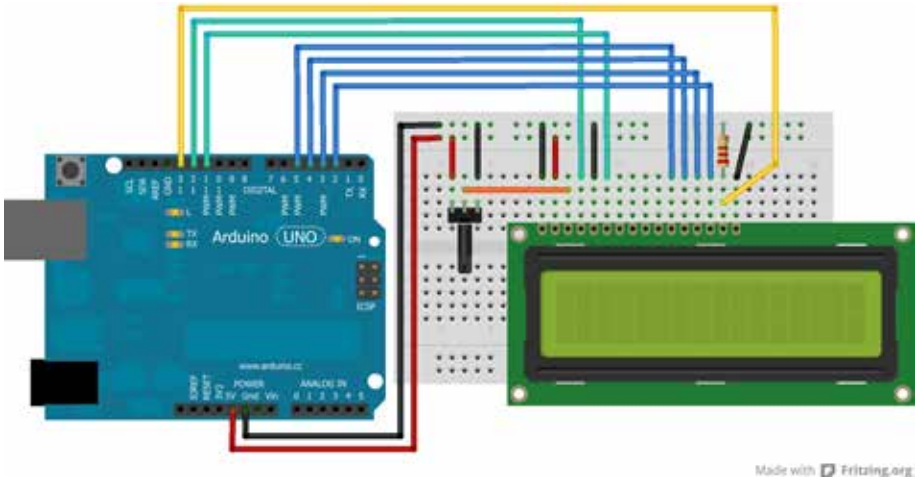
- LCD RS pin to digital pin 12
- LCD Enable pin to digital pin 11
- LCD D4 pin to digital pin 5
- LCD D5 pin to digital pin 4
- LCD D6 pin to digital pin 3
- LCD D7 pin to digital pin 2

Additionally, wire a 10K pot to +5V and GND, with it's wiper (output) to LCD screens VO pin (pin3).

## *SCHEMATIC*



*IMAGE*




*CODE*

```
/*
 LiquidCrystal Library - TextDirection

 Demonstrates the use a 16x2 LCD display.  The LiquidCrystal
 library works with all LCD displays that are compatible with the
 Hitachi HD44780 driver. There are many of them out there, and you
 can usually tell them by the 16-pin interface.

 This sketch demonstrates how to use leftToRight() and rightToLeft()
 to move the cursor.

 The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)

 Library originally added 18 Apr 2008
 by David A. Mellis
 library modified 5 Jul 2009
 by Limor Fried (http://www.ladyada.net)
 example added 9 Jul 2009
 by Tom Igoe
 modified 22 Nov 2010
 by Tom Igoe

 This example code is in the public domain.

 http://arduino.cc/en/Tutorial/LiquidCrystalTextDirection

 */

// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int thisChar = 'a';

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // turn on the cursor:
  lcd.cursor();
}

void loop() {
  // reverse directions at 'm':
  if (thisChar == 'm') {
    // go right for the next letter
    lcd.rightToLeft();
  }
  // reverse again at 's':
  if (thisChar == 's') {
    // go left for the next letter
    lcd.leftToRight();
  }
  // reset at 'z':
  if (thisChar > 'z') {
    // go to (0,0):
    lcd.home();
    // start again at 0
    thisChar = 'a';
  }
  // print the character
  lcd.write(thisChar);
  // wait a second:
  delay(1000);
  // increment the letter:
  thisChar++;
}
```

# Interface with a Display

*EXERCISE*

Use a display to augment interaction on your project

*HARDWARE REQUIRED*

Arduino Board
LED matrix or
LCD Character Display or
Nokia 5110 Displat

*SCHEMATIC*