UNC CHARLOTTE
ECGR 6185 FINAL PROJECT REPORT

STATUS BOARD

A REMOTE UPDATE STATUS DISPLAY BOARD FROM TEAM **INCREDIBLES!**

Team: Shruti Gangadhar
Sanket Shingte
Punit Salian
Dheeraj Suvarna

Date: May 10, 2016

8700 PHILLIPS RD
CHARLOTTE, NC 28223

"I will be out of office on May 15th. Leave the ECGR5101 HW3 under the door"

"If you have almost no food, you need a lot of beer to be happy! :)"

"Office hours: M & W from 2pm to 4pm"

"err, no, I will be there today, from 1:30 pm to 3:30pm."

Isn't it nice to have a status display board on our professors' office door that they can update remotely using their mobile phone application?

Well, here you go! We have the "STATUS BOARD" from team INCREDIBLES!

Contents:

Introduction:

“STATUS BOARD” is unique, easy-to-use product that consists of a display which can be mounted on an office door and have it display messages. The messages can be sent to the device remotely over the internet and the display updates the message almost instantly. This is a major improvement over having a white board and having to physically go there to erase the previously written message and update it with a new one.

The device itself consists of a 16x2 LCD Character Display, a microcontroller unit and a rechargeable battery. The display product comes with a custom built mobile phone application.

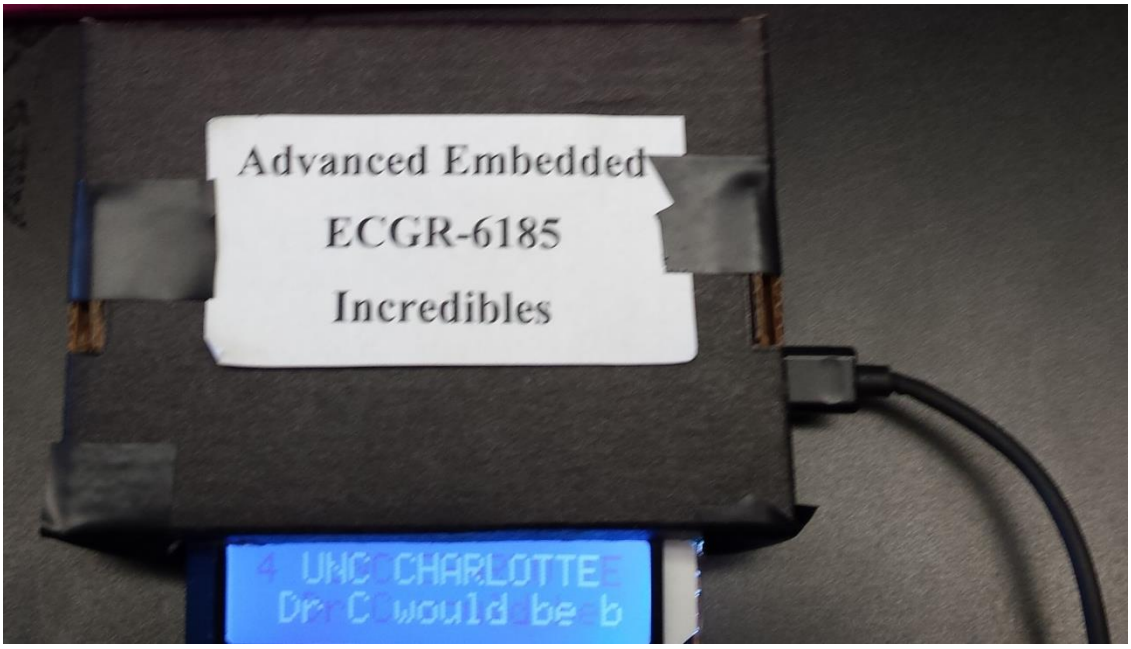


Fig 1. “Status Board” product

Working Principle:

The mobile app posts a message to the cloud. The cloud routes the message to the device based on “Device ID” and encryption is based on “Token”.

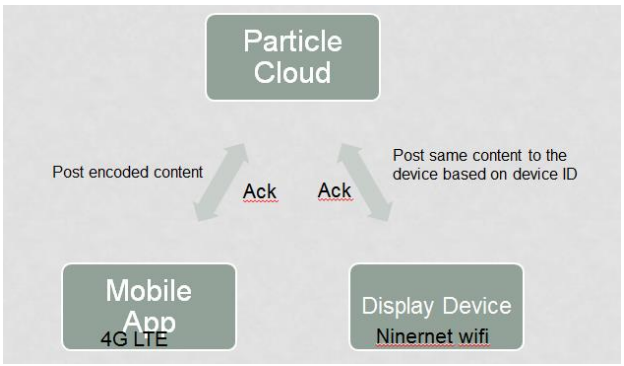


Fig 2. Functional Block Diagram

The crux of the device is the microcontroller unit that is combined with the Wi-Fi connectivity. The Particle Photon is chosen for this purpose. The photon device has 1Mb flash memory and 8 digital I/O pins making it easy to interface the LCD.

The mobile application posts the encrypted content to the URL embedded in the app and the particle cloud gets the content. Based on the Device ID and token, the cloud posts the same message to the Particle Device. The particle then displays the message on the LCD display.

The photon particle function called the spark function offers API to do the same. We have a function interfaced between the cloud and the particle device. The function accepts a string and sends an acknowledge signal.

Particle.function()

Expose a *function* through the Cloud so that it can be called with `POST /v1/devices/{DEVICE_ID}/{FUNCTION}`.

Up to 15 cloud functions may be registered and each function name is limited to a maximum of 12 characters.

In order to register a cloud function, the user provides the `funcKey`, which is the string name used to make a POST request and a `funcName`, which is the actual name of the function that gets called in your app. The cloud function can return any integer; `-1` is commonly used for a failed function call.

A cloud function is set up to take one argument of the String datatype. This argument length is limited to a max of 63 characters.

```
// SYNTAX
bool success = Particle.function("funcKey", funcName);

// Cloud functions must return int and take one String
int funcName(String extra) {
  return 0;
}









// EXAMPLE USAGE

int brewCoffee(String command);

void setup()
{
  // register the cloud function
  Particle.function("brew", brewCoffee);
}
```

Fig 3. Particle function

Device Usage – Operating Guide:

1. Give the power via USB. It takes 3.3 V as input supply.
2. Now display will show the name of professor and his or her office room number.
3. If now professor wants to update to the status; professor can enter to app.
4. In this mobile app professor can type the message he wants to display and press the send button.

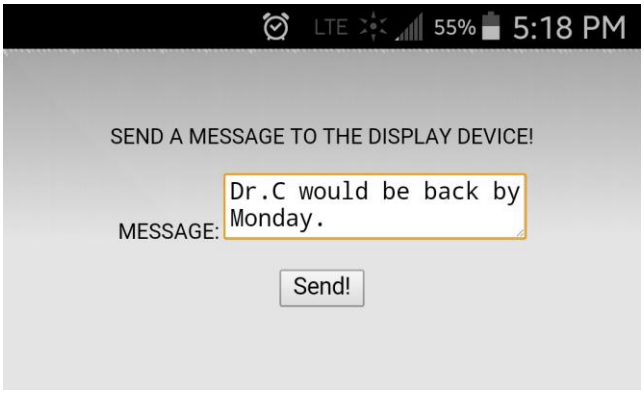


Fig 4. Mobile Application

5. Now we can see that the message will be displayed on the display.

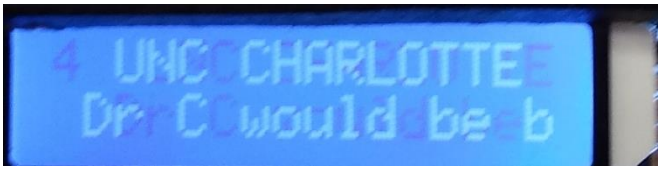


Fig 5. Message displayed on LCD

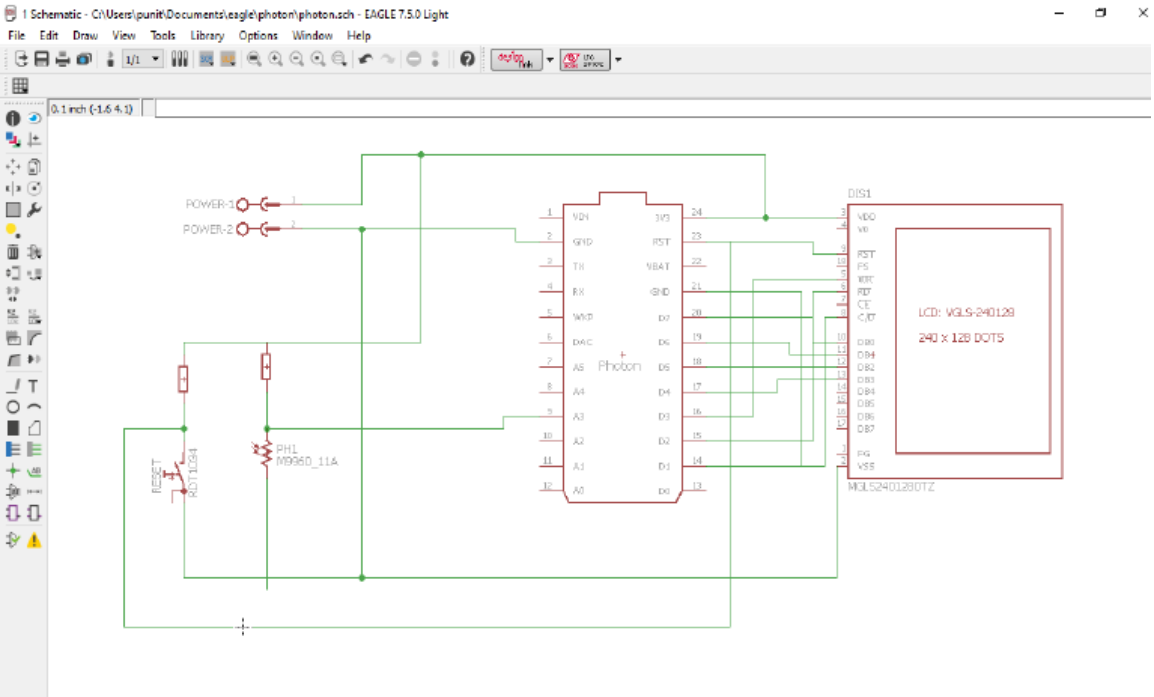
Usage Caution:

1. Do not drop the device.
2. Keep it away from moisture/water.
3. Do not provide voltage above 3.3V

Requirements List:

Requirements	Achieved
Small Size	Y
Light weight	Y
Instantaneous message update	Y
Mobile Application	Y
LCD Display 140 characters at once	N

Final Design Schematic



Final Bill of Material:

Product:

1. Box (Carton) – 10001
2. User Manual – 10002
3.. Battery Pack – 10003
4. Device – 10005
5. Door Mount – 10006

Device:

1. Mechanical Enclosure- 20001
2. Processor + Wifi module – Particle Photon – 20002
3. LCD Display – 20003
4. PCB – 20004
5. Connecting wires – 20005
6. Battery casing/connector – 20006

7. Battery – 20007

Bill of Materials (BOM):

Quantity	Part no	Description	Source
1	20002	Particle Photon	Adafruit.com
1	20003	16x2 Character LCD	Adafruit.com

For 1000 pieces:

1	Particle Photon	$19	$13.3
2	LCD	$9.95	$4.975
	Total	$28.95	$18.275

PURCHASE REQUISITION

Prepared

DATE: 4/27/16

BSS Recvd DATE:

FORWARD THIS FORM TO YOUR DEPARTMENTAL BSS / OTHER RESPONSIBLE PARTY

PURCHASE ORDER NO.	
REQUISITION NO.	
FUND / INDEX NUMBER	540168/
DEPARTMENT	Electrical and Computer Eng
BLDG. / ROOM	EPIC 2254
MARK FOR	Incredibles
TELEPHONE	704-687-8597

ITEM NO.	QUAN-TITY	UNIT	DESCRIPTION	UNIT PRICE	TOTAL PRICE
1	1	1	Particle Photon ID: 2721	19.00	19.00
2	1	1	Standard LCD 16x2 ID: 181	9.95	9.95
3		1			
4		1			
5		1			
6		1			
7		1			
8		1			
9					
10					

Funds for this purpose in the amount of order total will be reserved in the account(s) listed.

(If funded from more than one account, list accounts and amounts OR percentage by account.)

	Account	%		Amount
Acct 1				
Acct 2			OR	
Acct 3				

PG 1SUB-TOTAL 28.95

PG 2 SUB-TOTAL

Tax (if needed)

ORDER TOTAL 28.95

Suggested Vendors:

adafruit.com		

☑ Sole Source, attach justification on next sheet or attach as separate file.

Form attached to your email indicates your approval. If manually submitted, an authorized individual's signature is required:

Rev 4-05

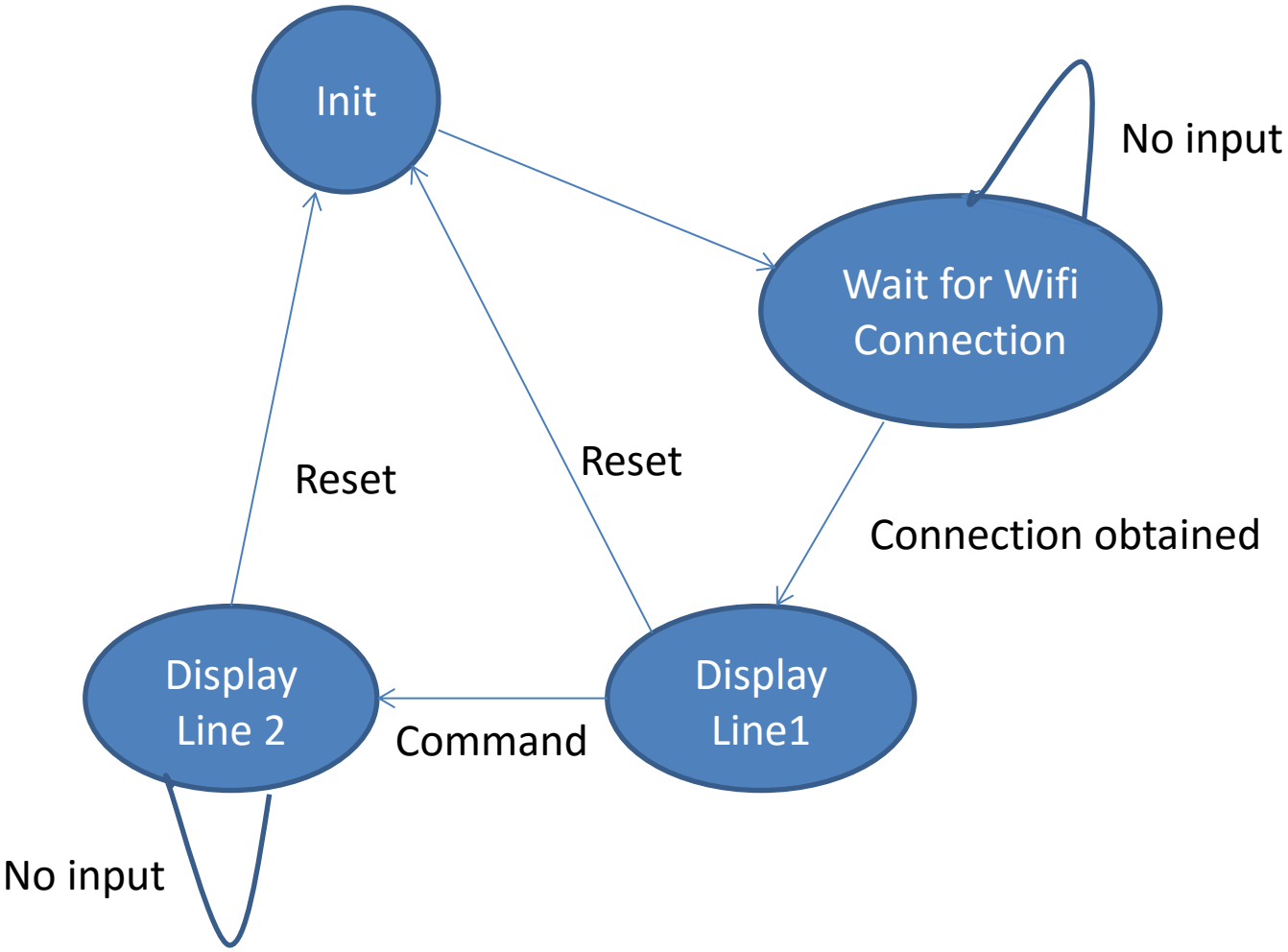
Device Testing:

Planning and Preparing to Perform Device Testing

Methodology and devices to facilitate usability testing on phones, tablets, and eReaders continue to develop. When planning mobile device testing, you should think about:

- **Your timeframe and budget**. Knowing these will help you to determine which processes and equipment will work best based your needs.
- **The physical setup of the space and how you will capture the test.** This may range from a lower-fidelity arrangement to using a specialized platform and the software.
- **What's included in your screener.** It's important to be specific about:
  - The type of phone the participant must have
  - What generation of phone, if necessary.
  - According to that you have to develop the mobile application.
- **How the test is structured**?
- **Consider how frequently you will need to perform testing**.
- **Ensure there is connectivity in the space.** Make sure that there is a good cross section of wireless carriers getting service in the area you intend to test. Connectivity is not always universal and you must pilot test before your participants arrive.
- **Regarding our product:** It is very easy to test the electronic nameplate. We just need to provide 3.3 V to the device. It then turns on the backlight. As we will show in the video we will make the mobile application on and just have send the message. This message will appear on the screen and we can make sure that our device is not working properly. If it is not working fine there will be some points for trouble shooting like Wi-Fi connection, USB input power supply etc.

State Machine:



When we power on the device, it goes into the init state. It waits on the internet connection. Here, the device is connected to UNCC Niner net wifi. When the device gets the connection, it just displays the Line 1 of LCD and waits. Once the user updates a message on the app and sends it, the device receives the message and that gets updated on Line 2 of LCD.

Algorithm:

SETUP()

INITIALIZE ALL HARDWARE AND DISPLAY

LOOP()

FOREVER LOOP WAITING FOR MESSAGE VIA MOBILE APPLICATION

MESSAGEHANDLER()

THIS HANDLER IS ACTIVATED AS SOON AS THE MESSAGE IS SENT FROM THE CLOUD

CODE LISTING

The following is the main code:

```
#include "application.h"

/*
  The circuit:
 * LCD RS pin to digital pin D0
 * LCD EN pin to digital pin D1
 * LCD D4 pin to digital pin D2
 * LCD D5 pin to digital pin D3
 * LCD D6 pin to digital pin D4
 * LCD D7 pin to digital pin D5
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)
 * connect R/W (pin 5) to ground
 */

#include "LiquidCrystal/LiquidCrystal.h"

LiquidCrystal lcd(D0, D1, D2, D3, D4, D5);

//pin variable
int led2 = D7;

//temp variable to values
long c;
String s1,s2;
int state=0;
int flag=0;
//intialize hardware here and initial setting
void setup() {

lcd.begin(16,2);// set up the LCD's number of columns and rows:
lcd.setCursor(1, 0);
lcd.print("UNC CHARLOTTE");

   Spark.function("led",ledToggle);
}
```

```
void loop() {

lcd.scrollDisplayLeft();

delay(500);

}


int ledToggle(String command) {
   /* Spark.functions always take a string as an argument and return an
integer.
   Since we can pass a string, it means that we can give the program
commands on how the function should be used.
   In this case, we are sending the string “command” to lcd.print function
so that it gets displayed on the LCD. */

lcd.setCursor(0, 1);
lcd.print( command);


}
```

The LiquidCrystal.h contains all the function details related to LCD.

```
#ifndef LiquidCrystal_h
#define LiquidCrystal_h

// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00
```

```
// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

class LiquidCrystal : public Print {
public:
  LiquidCrystal(uint8_t rs, uint8_t enable,
                uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7);
  LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7);
  LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3);
  LiquidCrystal(uint8_t rs, uint8_t enable,
                uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3);

void init(uint8_t fourbitmode, uint8_t rs, uint8_t rw, uint8_t enable,
          uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
          uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7);

  void begin(uint8_t cols, uint8_t rows, uint8_t charsize = LCD_5x8DOTS);

void clear();
  void home();
```

```
void noDisplay();
  void display();
  void noBlink();
  void blink();
  void noCursor();
  void cursor();
  void scrollDisplayLeft();
  void scrollDisplayRight();
  void leftToRight();
  void rightToLeft();
  void autoscroll();
  void noAutoscroll();

  void createChar(uint8_t, uint8_t[]);
  void setCursor(uint8_t, uint8_t);
  virtual size_t write(uint8_t);
  void command(uint8_t);
private:
  void send(uint8_t, uint8_t);
  void write4bits(uint8_t);
  void write8bits(uint8_t);
  void pulseEnable();

  uint8_t _rs_pin; // LOW: command.  HIGH: character.
  uint8_t _rw_pin; // LOW: write to LCD.  HIGH: read from LCD.
  uint8_t _enable_pin; // activated by a HIGH pulse.
  uint8_t _data_pins[8];

  uint8_t _displayfunction;
  uint8_t _displaycontrol;
  uint8_t _displaymode;

  uint8_t _initialized;

  uint8_t _numlines,_currline;
};

#endif
```

The LiquidCrystal.cpp has the LCD related functions.

```
#include "application.h"

#include "LiquidCrystal.h"

// When the display powers up, it is configured as follows:
//
// 1. Display clear
// 2. Function set:
//    DL = 1; 8-bit interface data
//    N = 0; 1-line display
//    F = 0; 5x8 dot character font
// 3. Display on/off control:
//    D = 0; Display off
//    C = 0; Cursor off
//    B = 0; Blinking off
// 4. Entry mode set:
//    I/D = 1; Increment by 1
//    S = 0; No shift
//
// Note, however, that resetting the Arduino doesn't reset the LCD, so we
// can't assume that its in that state when a sketch starts (and the
// LiquidCrystal constructor is called).

LiquidCrystal::LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                             uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                             uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7)
{
  init(0, rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7);
}

LiquidCrystal::LiquidCrystal(uint8_t rs, uint8_t enable,
                             uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                             uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7)
{
  init(0, rs, 255, enable, d0, d1, d2, d3, d4, d5, d6, d7);
}

LiquidCrystal::LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
                             uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3)
{
  init(1, rs, rw, enable, d0, d1, d2, d3, 0, 0, 0, 0);
}
```

```
LiquidCrystal::LiquidCrystal(uint8_t rs,  uint8_t enable,
                             uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3)
{
  init(1, rs, 255, enable, d0, d1, d2, d3, 0, 0, 0, 0);
}

void LiquidCrystal::init(uint8_t fourbitmode, uint8_t rs, uint8_t rw, uint8_t enable,
                         uint8_t d0, uint8_t d1, uint8_t d2, uint8_t d3,
                         uint8_t d4, uint8_t d5, uint8_t d6, uint8_t d7)
{
  _rs_pin = rs;
  _rw_pin = rw;
  _enable_pin = enable;

  _data_pins[0] = d0;
  _data_pins[1] = d1;
  _data_pins[2] = d2;
  _data_pins[3] = d3;
  _data_pins[4] = d4;
  _data_pins[5] = d5;
  _data_pins[6] = d6;
  _data_pins[7] = d7;

  // Always 4-bit mode, don't waste pins!
  //
  //if (fourbitmode)
    _displayfunction = LCD_4BITMODE | LCD_1LINE | LCD_5x8DOTS;
  //else
    //_displayfunction = LCD_8BITMODE | LCD_1LINE | LCD_5x8DOTS;
}

void LiquidCrystal::begin(uint8_t cols, uint8_t lines, uint8_t dotsize) {
  if (lines > 1) {
    _displayfunction |= LCD_2LINE;
  }
  _numlines = lines;
  _currline = 0;

pinMode(_rs_pin, OUTPUT);
  // we can save 1 pin by not using RW. Indicate by passing 255 instead of pin#
  if (_rw_pin != 255) {
    pinMode(_rw_pin, OUTPUT);
  }
  pinMode(_enable_pin, OUTPUT);
```

```
// for some 1 line displays you can select a 10 pixel high font
  if ((dotsize != 0) && (lines == 1)) {
    _displayfunction |= LCD_5x10DOTS;
  }

   // according to datasheet, we need at least 40ms after power rises above 2.7V
  // before sending commands. Arduino can turn on way befer 4.5V so we'll wait 50
  delayMicroseconds(50000);
  // Now we pull both RS and R/W low to begin commands
  digitalWrite(_rs_pin, LOW);
  digitalWrite(_enable_pin, LOW);
  if (_rw_pin != 255) {
    digitalWrite(_rw_pin, LOW);
  }

  // 4-Bit initialization sequence from Technobly
  write4bits(0x03);         // Put back into 8-bit mode
  delayMicroseconds(5000);

  write4bits(0x08);         // Comment this out for V1 OLED
  delayMicroseconds(5000);  // Comment this out for V1 OLED

  write4bits(0x02);         // Put into 4-bit mode
  delayMicroseconds(5000);
  write4bits(0x02);
  delayMicroseconds(5000);
  write4bits(0x08);
  delayMicroseconds(5000);

  command(LCD_DISPLAYCONTROL);                // Turn Off
  delayMicroseconds(5000);
  command(LCD_FUNCTIONSET | _displayfunction);  // Set # lines, font size, etc.
  delayMicroseconds(5000);
  clear();                                    // Clear Display
  command(LCD_ENTRYMODESET | LCD_ENTRYLEFT);    // Set Entry Mode
  delayMicroseconds(5000);
  home();                                     // Home Cursor
  delayMicroseconds(5000);
  command(LCD_DISPLAYCONTROL | LCD_DISPLAYON);  // Turn On -
enable cursor & blink
  delayMicroseconds(5000);
 }
```

```
/********** high level commands, for the user! */
void LiquidCrystal::clear()
{
  command(LCD_CLEARDISPLAY);  // clear display, set cursor position to zero
  delayMicroseconds(5000);  // this command takes a long time!
}

void LiquidCrystal::home()
{
  command(LCD_RETURNHOME);  // set cursor position to zero
  delayMicroseconds(5000);  // this command takes a long time!
}

void LiquidCrystal::setCursor(uint8_t col, uint8_t row)
{
  int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 };
  if ( row > _numlines ) {
    row = _numlines-1;    // we count rows starting w/0
  }
  command(LCD_SETDDRAMADDR | (col + row_offsets[row]));
}

// Turn the display on/off (quickly)
void LiquidCrystal::noDisplay() {
  _displaycontrol &= ~LCD_DISPLAYON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}
void LiquidCrystal::display() {
  _displaycontrol |= LCD_DISPLAYON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}


// Turns the underline cursor on/off
void LiquidCrystal::noCursor() {
  _displaycontrol &= ~LCD_CURSORON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}
void LiquidCrystal::cursor() {
  _displaycontrol |= LCD_CURSORON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}
```

```
// Turn on and off the blinking cursor
void LiquidCrystal::noBlink() {
  _displaycontrol &= ~LCD_BLINKON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}
void LiquidCrystal::blink() {
  _displaycontrol |= LCD_BLINKON;
  command(LCD_DISPLAYCONTROL | _displaycontrol);
}

// These commands scroll the display without changing the RAM
void LiquidCrystal::scrollDisplayLeft(void) {
  command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT);
}
void LiquidCrystal::scrollDisplayRight(void) {
  command(LCD_CURSORSHIFT | LCD_DISPLAYMOVE |
LCD_MOVERIGHT);
}

// This is for text that flows Left to Right
void LiquidCrystal::leftToRight(void) {
  _displaymode |= LCD_ENTRYLEFT;
  command(LCD_ENTRYMODESET | _displaymode);
}

// This is for text that flows Right to Left
void LiquidCrystal::rightToLeft(void) {
  _displaymode &= ~LCD_ENTRYLEFT;
  command(LCD_ENTRYMODESET | _displaymode);
}

// This will 'right justify' text from the cursor
void LiquidCrystal::autoscroll(void) {
  _displaymode |= LCD_ENTRYSHIFTINCREMENT;
  command(LCD_ENTRYMODESET | _displaymode);
}

// This will 'left justify' text from the cursor
void LiquidCrystal::noAutoscroll(void) {
  _displaymode &= ~LCD_ENTRYSHIFTINCREMENT;
  command(LCD_ENTRYMODESET | _displaymode);
}
```

```
// Allows us to fill the first 8 CGRAM locations
// with custom characters
void LiquidCrystal::createChar(uint8_t location, uint8_t charmap[]) {
  location &= 0x7; // we only have 8 locations 0-7
  command(LCD_SETCGRAMADDR | (location << 3));
  for (int i=0; i<8; i++) {
    write(charmap[i]);
  }
}

/*********** mid level commands, for sending data/cmds */

inline void LiquidCrystal::command(uint8_t value) {
  send(value, LOW);
}

inline size_t LiquidCrystal::write(uint8_t value) {
  send(value, HIGH);
  return 1;
}

/************ low level data pushing commands **********/

// write either command or data, with automatic 4/8-bit selection
void LiquidCrystal::send(uint8_t value, uint8_t mode) {
  digitalWrite(_rs_pin, mode);

  // if there is a RW pin indicated, set it low to Write
  if (_rw_pin != 255) {
    digitalWrite(_rw_pin, LOW);
  }

  if (_displayfunction & LCD_8BITMODE) {
    write8bits(value);
  } else {
    write4bits(value>>4);
    write4bits(value);
  }
}

void LiquidCrystal::pulseEnable(void) {
  digitalWrite(_enable_pin, LOW);
  delayMicroseconds(1);
  digitalWrite(_enable_pin, HIGH);
  delayMicroseconds(1);    // enable pulse must be >450ns
  digitalWrite(_enable_pin, LOW);
  delayMicroseconds(100);   // commands need > 37us to settle
}
```

```
void LiquidCrystal::write4bits(uint8_t value) {
  for (int i = 0; i < 4; i++) {
    pinMode(_data_pins[i], OUTPUT);
    digitalWrite(_data_pins[i], (value >> i) & 0x01);
  }
  pulseEnable();
}

void LiquidCrystal::write8bits(uint8_t value) {
  for (int i = 0; i < 8; i++) {
    pinMode(_data_pins[i], OUTPUT);
    digitalWrite(_data_pins[i], (value >> i) & 0x01);
  }
  pulseEnable();
}
```