תצוגת גביש נוזלי 5110

א. מאפיינים

ב. 48 שורות על 84 עמודות של נקודות (פיקסלים).

ג. ממשק טורי עם תקשורת מקסימלית של 4 מגה ביט בשנייה (תקשורת SPI).

ד. בקר פנימי PCD8544

ה. תאורת רקע של לד

ו. מתח הפעלה מ 2.7 עד 5 וולט

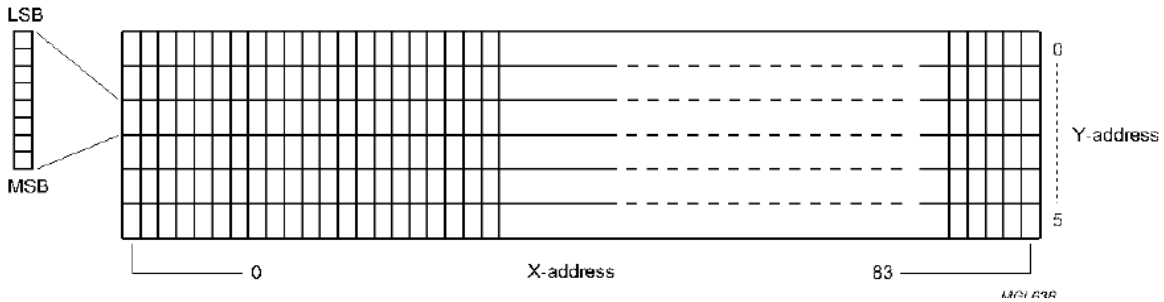
ז. תצרוכת הספק נמוכה . מתאים לפעולה עם סוללה.

ח. תומך בכניסות CMOS .

ט. טמפרטורת פעולה -25 עד 70 מעלות צלסיוס.

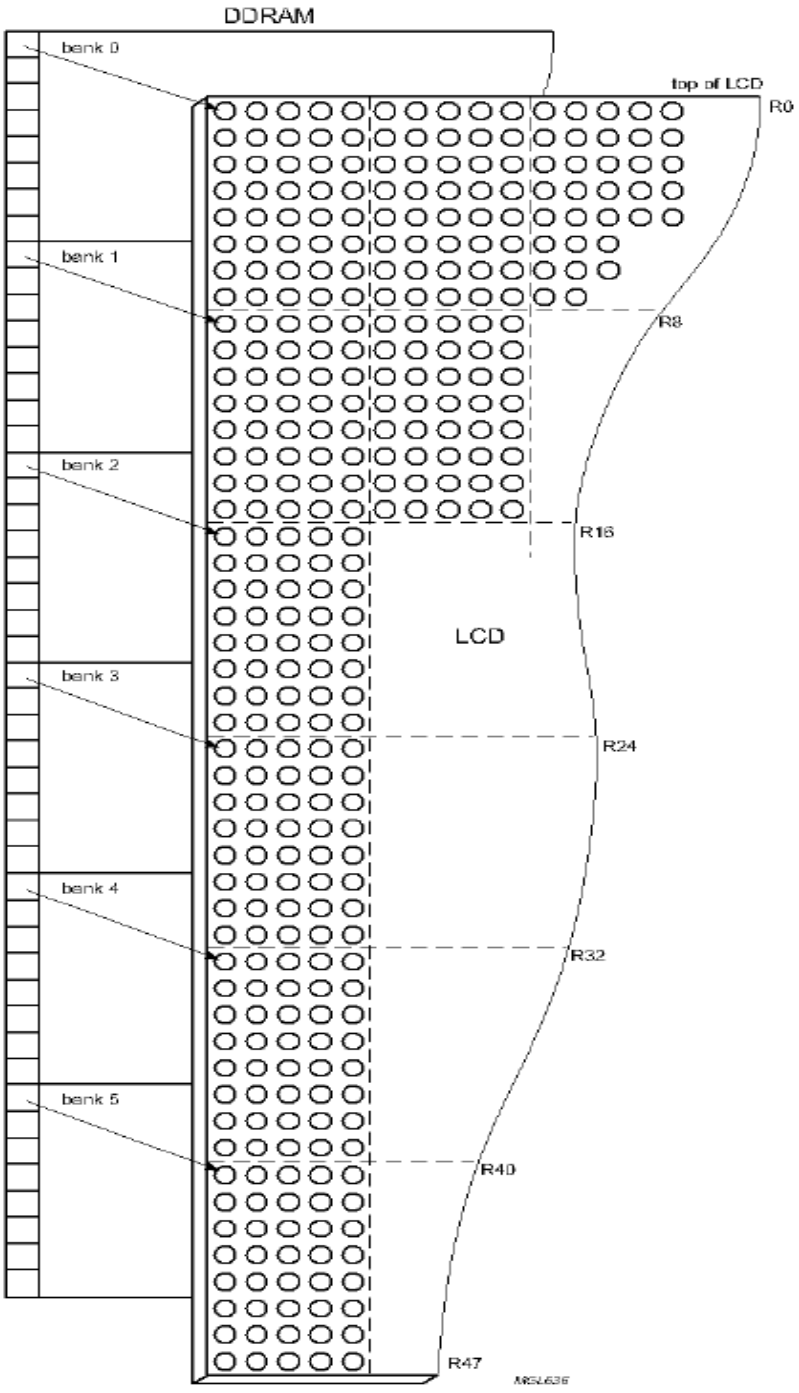
ב. מיעון כתובות

ארגון הכתובות בזיכרון ה DDRAM הוא מטריצה של 6 שורות) כתובות (Y מכתובת Y של 0 עד כתובת Y של 5 ו 84עמודות מ x=0 ועד x=83 . בציר Y יש 6 בנקים כל בנק של 8 ביט . באיור 1 מתוארים 6 הבנקים של ציר Y .



איור 1 : 6 הבנקים של ציר Y

באיור 2 ניתן לראות את מבנה השורות והעמודות של הזיכרון.

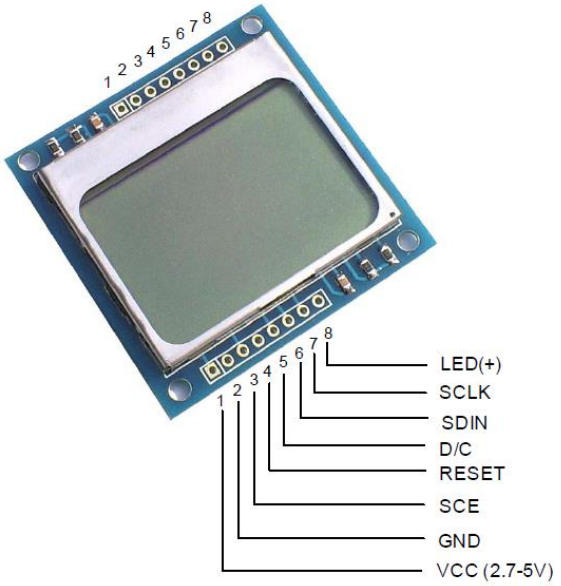


איור 2 : מבנה השורות והעמודות של הזיכרון

ניתן לכתוב נתונים לכתובות הזיכרון DDRAM ברציפות וערכי X ו Y יקודמו אוטומטית . במקרה כזה ישנן 2שיטות לקבוע את פורמט הפעולה של הכתובת .הראשונה אופן מיעון אנכי V=1 שבו נוסף 1 בכתובת ה Y כל פעם) ציור .(3 אופן שני הוא מיעון אופקי שבו נוסף 1 כל פעם לכתובת ה) X ציור.(4

ג. חיבור ובקרה של התצוגה

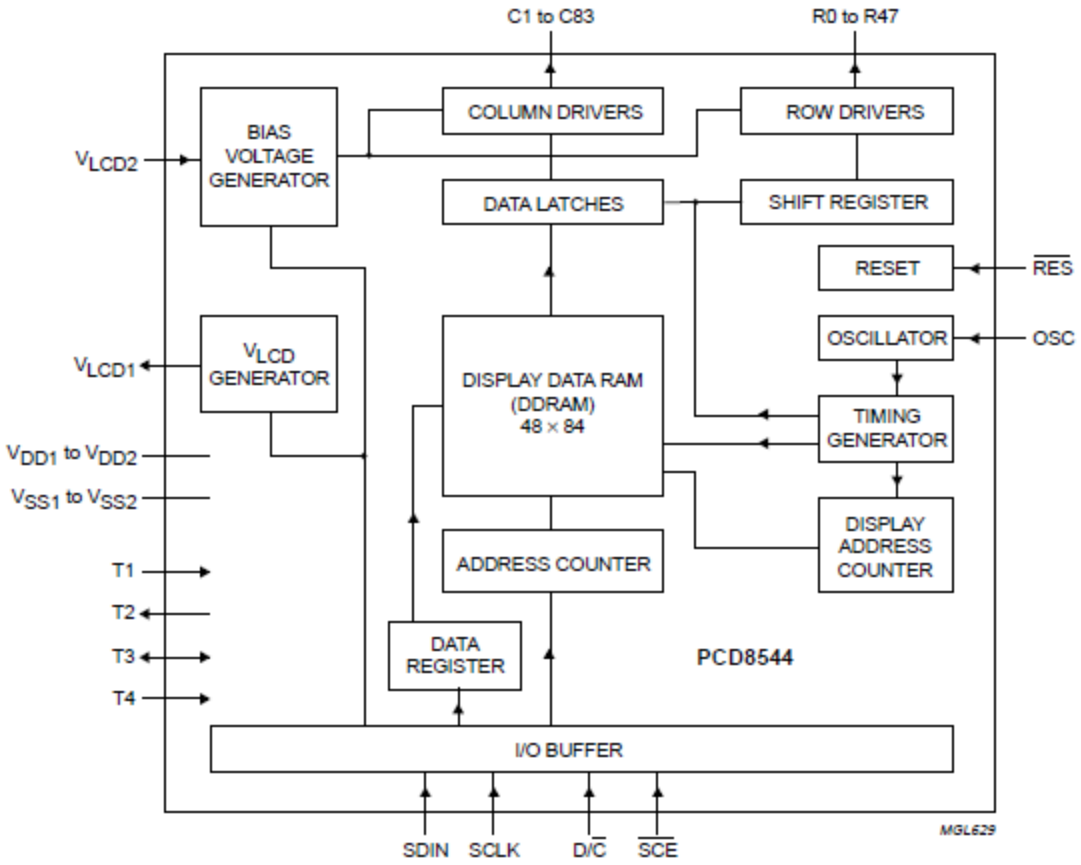
באיור 3 רואים את התצוגה ומיקום ההדקים שלה.



איור 3 : התצוגה וההדקים

1. Vcc - מתח ספק מ 2.7 עד 5 וולט .
2. GND - הדק האדמה.
3. SCE - Chip Select בחירת הרכיב . (פעיל בנמוך). מתאים ל SS של תקשורת SPI .
4. RESET - אות האיפוס של הרכיב
5. D/C - האם שולחים פקודה (0) או נתון.(1)
6. SDIN - Serial Data In - כניסת נתון טורית . מתאים להדק ה MOSI בתקשורת SPI .
7. SCLK - Serial ClocK שעון טורי .פולסי השעון הטורי המסנכרן את הנתון הטורי לתוך הרכיב. מתאים להדק SCLK בתקשורת SPI .
8. LED – הדק להפעלת תאורה אחורית

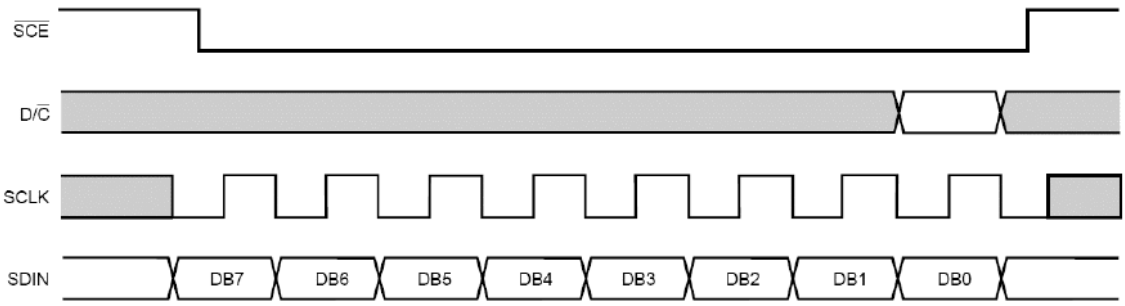
ד. מבנה פנימי של בקר התצוגה PCD8544



איור 4 : מבנה פנימי של הבקר **PCD8544**

ה. שידור טורי של פקודה/נתון

באיור 5 מתואר שידור טורי של פקודה / נתון בתקשורת טורית SPI .



איור 5 : שידור של בייט בתקשורת טורית

ה.1 השידור מתחיל תמיד עם הורדת הדק $\overline{SCE}$ ל 0 . (מקביל ל SS בתקשורת SPI).

ה.2 אם שולחים פקודה אז שמים בהדק C/D 0 ואם שולחים נתון שמים בהדק 1 .

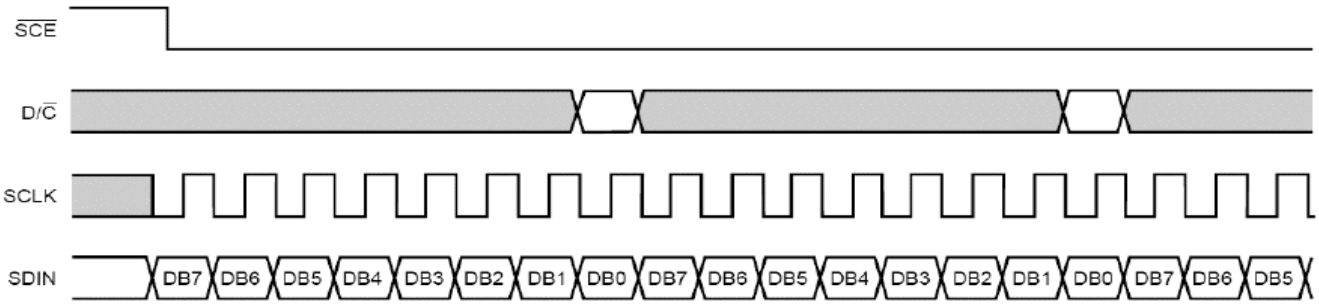
ה.3 לאחר מכן שמים את ביט ה MSB על הקו SDIN (מקביל לקו MOSI ב SPI) .

ה.4 מעלים את הדק ה SCLK ל 1 ואז הנתון מסונכרן אל הרכיב.

ה.5 כך ממשיכים את הביטים אחד אחרי השני כאשר בכל עליית שעון הביט מסונכרן לתוך הרכיב.

ה.6 בסיום הכנסת ביט ה LSB מעלים את הדק $\overline{SCE}$ ל 1 והנתון ננעל ברכיב.

כאשר רוצים לשדר בייט אחרי בייט אין צורך להעלות את הדק ה $\overline{SCE}$ אחרי כל בייט אלא רק בסיום השידור של כל הבייטים. הדבר מתואר באיור 6 .



איור 6 : שידור של מספר בייטים

ו. סט הפקודות

בטבלה 1 רואים את סט הפקודות של התצוגה

INSTRUCTION	D/$\overline{C}$	COMMAND BYTE								DESCRIPTION
		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
(H = 0 or 1)										
NOP	0	0	0	0	0	0	0	0	0	no operation
Function set	0	0	0	1	0	0	PD	V	H	power down control; entry mode; extended instruction set control (H)
Write data	1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	writes data to display RAM
(H = 0)										
Reserved	0	0	0	0	0	0	1	X	X	do not use
Display control	0	0	0	0	0	1	D	0	E	sets display configuration
Reserved	0	0	0	0	1	X	X	X	X	do not use
Set Y address of RAM	0	0	1	0	0	0	Y_2	Y_1	Y_0	sets Y-address of RAM; $0 \le Y \le 5$
Set X address of RAM	0	1	X_6	X_5	X_4	X_3	X_2	X_1	X_0	sets X-address part of RAM; $0 \le X \le 83$
(H = 1)										
Reserved	0	0	0	0	0	0	0	0	1	do not use
	0	0	0	0	0	0	0	1	X	do not use
Temperature control	0	0	0	0	0	0	1	TC_1	TC_0	set Temperature Coefficient (TC_x)
Reserved	0	0	0	0	0	1	X	X	X	do not use
Bias system	0	0	0	0	1	0	BS_2	BS_1	BS_0	set Bias System (BS_x)
Reserved	0	0	1	X	X	X	X	X	X	do not use
Set V_{OP}	0	1	V_{OP6}	V_{OP5}	V_{OP4}	V_{OP3}	V_{OP2}	V_{OP1}	V_{OP0}	write V_{OP} to register

בטבלה 2 יש הסברים עבור טבלה 1

BIT	0	1
PD	chip is active	chip is in Power-down mode
V	horizontal addressing	vertical addressing
H	use basic instruction set	use extended instruction set
D and E		
00	display blank	
10	normal mode	
01	all display segments on	
11	inverse video mode	
TC_1 and TC_0		
00	V_{LCD} temperature coefficient 0	
01	V_{LCD} temperature coefficient 1	
10	V_{LCD} temperature coefficient 2	
11	V_{LCD} temperature coefficient 3	

טבלה 2 : הסבר לטבלה 1

ז. פונקציות של התצוגה

לתצוגה יש ספרייה הנקראת . ADAFRUIT_PCD8544 הספרייה נכתבה על ידי
Limor Fried/Ladyada עבורAdafruit Industries .

https://github.com/adafruit/Adafruit-PCD8544-Nokia-5110-LCD-library

ספריה נוספת היא מהאתר http://playground.arduino.cc/Code/PCD8544

נזכיר חלק מהפונקציות של הספרייה:

begin(uint8_t contrast, uint8_t bias);

הפונקציה מאתחלת את התצוגה .וקובעת את הקונטרסט והממתח . היא מזמנת 4 פונקציות:

SPI.begin;( )

אתחול הדקי תקשורת ה SPI

SPI.setClockDivider(PCD8544_SPI_CLOCK_DIV);

קביעת קצב התקשורת של SPI

SPI.setDataMode(SPI_MODE0);

קביעת אופן 0 של התקשורת.

SPI.setBitOrder(MSBFIRST);

קובעים שהשידור מהביט הגבוה אל הנמוך (בדרך כלל בתקשורת SPI הוא מהנמוך אל הגבוה) .

clearDisplay(void)

ניקוי התצוגה - ניקוי ה DDRAM .

setContrast(uint8_t val);

כיוון הקונטרסט – (ניגודיות) לערך הרצוי.

setCursor(x , y);

הבאת הסמן למקום הרצוי.

drawPixel(int16_t x, int16_t y, uint16_t color)

ציור נקודה(פיקסל) במיקום X ו Y בצבע רצוי.

data(uint8_t c);

כתיבת נתון לתצוגה

command(uint8_t c);

כתיבת פקודה לתצוגה

display();

מציגה את מה שיש ב DDRAM במסך. כאשר מדפיסים לתצוגה מחרוזת או כל דבר אחר זה נכתב ב DDRAM ורק עם הפונקציה display זה יוצג בתצוגה !!

print(" string ");

מדפיסה את המחרוזת שבין הגרשיים בתצוגה

print(משתנה) ;

מדפיסה את הערך של המשתנה בתצוגה

drawLine(uint16_t x1,uint16_t y1 , uint16_t x2,uint16_t y2, uint16_t color);

ציור של קו בין נקודות x1 y1 לנקודות x2 y2 בצבע הרצוי .

הפונקציות מהאתר של github

https://github.com/adafruit/Adafruit-PCD8544-Nokia-5110-LCD-library

371 lines (298 sloc) 10.5 KB

```
/*********************************************************************
This is a library 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 below must be included in any redistribution

*********************************************************************/

//#include <Wire.h>

#include <avr/pgmspace.h>

#if defined(ARDUINO) && ARDUINO >= 100

  #include "Arduino.h"

#else

  #include "WProgram.h"

#endif

#ifdef __AVR__

  #include <util/delay.h>
```

```
#endif

#ifndef _BV

  #define _BV(x) (1 << (x))

#endif

#include <stdlib.h>

#include <Adafruit_GFX.h>

#include "Adafruit_PCD8544.h"

#ifndef _BV

  #define _BV(bit) (1<<(bit))

#endif

// the memory buffer for the LCD

uint8_t pcd8544_buffer[LCDWIDTH * LCDHEIGHT / 8] = {

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0xE0, 0xF0, 0xF8, 0xFC, 0xFC, 0xFE, 0xFF,
0xFC, 0xE0,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8,
0xF8, 0xF8,

0xF8, 0xF0, 0xF0, 0xE0, 0xE0, 0xC0, 0x80, 0xC0, 0xFC, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F,
0x3F, 0x7F,

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x1F, 0x3F, 0x7F,
0xFF, 0xFF,

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xE7, 0xC7, 0xC7, 0x87, 0x8F, 0x9F, 0x9F, 0xFF,
0xFF, 0xFF,

0xC1, 0xC0, 0xE0, 0xFC, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFC, 0xFC, 0xFC, 0xFC, 0xFE,
0xFE, 0xFE,

0xFC, 0xFC, 0xF8, 0xF8, 0xF0, 0xE0, 0xC0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x80, 0xC0, 0xE0, 0xF1, 0xFB, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x1F, 0x0F,
0x0F, 0x87,

0xE7, 0xFF, 0xFF, 0xFF, 0x1F, 0x1F, 0x3F, 0xF9, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8, 0xF8,
0xFD, 0xFF,

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x3F, 0x0F, 0x07, 0x01, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0xF0, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFE,

0x7E, 0x3F, 0x3F, 0x0F, 0x1F, 0xFF, 0xFF, 0xFF, 0xFC, 0xF0, 0xE0, 0xF1, 0xFF, 0xFF,
0xFF, 0xFF,

0xFF, 0xFC, 0xF0, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x01,

0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x0F, 0x1F, 0x3F,
0x7F, 0x7F,

0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x7F, 0x1F, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

};

// reduces how much is refreshed, which speeds it up!

// originally derived from Steve Evans/JCW's mod but cleaned up and

// optimized

//#define enablePartialUpdate

#ifdef enablePartialUpdate

static uint8_t xUpdateMin, xUpdateMax, yUpdateMin, yUpdateMax;

#endif

static void updateBoundingBox(uint8_t xmin, uint8_t ymin, uint8_t xmax, uint8_t
ymax) {

#ifdef enablePartialUpdate

  if (xmin < xUpdateMin) xUpdateMin = xmin;

  if (xmax > xUpdateMax) xUpdateMax = xmax;

  if (ymin < yUpdateMin) yUpdateMin = ymin;

  if (ymax > yUpdateMax) yUpdateMax = ymax;

#endif

}

Adafruit_PCD8544::Adafruit_PCD8544(int8_t SCLK, int8_t DIN, int8_t DC,

    int8_t CS, int8_t RST) : Adafruit_GFX(LCDWIDTH, LCDHEIGHT) {
```

```
  _din = DIN;
  _sclk = SCLK;
  _dc = DC;
  _rst = RST;
  _cs = CS;
}
Adafruit_PCD8544::Adafruit_PCD8544(int8_t SCLK, int8_t DIN, int8_t DC,
    int8_t RST) : Adafruit_GFX(LCDWIDTH, LCDHEIGHT) {
  _din = DIN;
  _sclk = SCLK;
  _dc = DC;
  _rst = RST;
  _cs = -1;
}
Adafruit_PCD8544::Adafruit_PCD8544(int8_t DC, int8_t CS, int8_t RST):
  Adafruit_GFX(LCDWIDTH, LCDHEIGHT) {
  // -1 for din and sclk specify using hardware SPI
  _din = -1;
  _sclk = -1;
  _dc = DC;
  _rst = RST;
  _cs = CS;
}
// the most basic function, set a single pixel
void Adafruit_PCD8544::drawPixel(int16_t x, int16_t y, uint16_t color) {
  if ((x < 0) || (x >= _width) || (y < 0) || (y >= _height))
    return;
```

```
  int16_t t;
  switch(rotation){
    case 1:
      t = x;
      x = y;
      y =  LCDHEIGHT - 1 - t;
      break;
    case 2:
      x = LCDWIDTH - 1 - x;
      y = LCDHEIGHT - 1 - y;
      break;
    case 3:
      t = x;
      x = LCDWIDTH - 1 - y;
      y = t;
      break;
  }
  if ((x < 0) || (x >= LCDWIDTH) || (y < 0) || (y >= LCDHEIGHT))
    return;
  // x is which column
  if (color)
    pcd8544_buffer[x+ (y/8)*LCDWIDTH] |= _BV(y%8);
  else
    pcd8544_buffer[x+ (y/8)*LCDWIDTH] &= ~_BV(y%8);
  updateBoundingBox(x,y,x,y);
}
// the most basic function, get a single pixel
```

```
uint8_t Adafruit_PCD8544::getPixel(int8_t x, int8_t y) {
  if ((x < 0) || (x >= LCDWIDTH) || (y < 0) || (y >= LCDHEIGHT))
    return 0;
  return (pcd8544_buffer[x+ (y/8)*LCDWIDTH] >> (y%8)) & 0x1;
}
void Adafruit_PCD8544::begin(uint8_t contrast, uint8_t bias) {
  if (isHardwareSPI()) {
    // Setup hardware SPI.
    SPI.begin();
    SPI.setClockDivider(PCD8544_SPI_CLOCK_DIV);
    SPI.setDataMode(SPI_MODE0);
    SPI.setBitOrder(MSBFIRST);
  }
  else {
    // Setup software SPI.
    // Set software SPI specific pin outputs.
    pinMode(_din, OUTPUT);
    pinMode(_sclk, OUTPUT);
    // Set software SPI ports and masks.
    clkport     = portOutputRegister(digitalPinToPort(_sclk));
    clkpinmask  = digitalPinToBitMask(_sclk);
    mosiport    = portOutputRegister(digitalPinToPort(_din));
    mosipinmask = digitalPinToBitMask(_din);
  }
  // Set common pin outputs.
  pinMode(_dc, OUTPUT);
  if (_rst > 0)
```

```
    pinMode(_rst, OUTPUT);
if (_cs > 0)
    pinMode(_cs, OUTPUT);
// toggle RST low to reset
if (_rst > 0) {
  digitalWrite(_rst, LOW);
  delay(500);
  digitalWrite(_rst, HIGH);
}
// get into the EXTENDED mode!
command(PCD8544_FUNCTIONSET | PCD8544_EXTENDEDINSTRUCTION );
// LCD bias select (4 is optimal?)
command(PCD8544_SETBIAS | bias);
// set VOP
if (contrast > 0x7f)
  contrast = 0x7f;
command( PCD8544_SETVOP | contrast); // Experimentally determined
// normal mode
command(PCD8544_FUNCTIONSET);
// Set display to Normal
command(PCD8544_DISPLAYCONTROL | PCD8544_DISPLAYNORMAL);
// initial display line
// set page address
// set column address
// write display data
// set up a bounding box for screen updates
updateBoundingBox(0, 0, LCDWIDTH-1, LCDHEIGHT-1);
```

```
  // Push out pcd8544_buffer to the Display (will show the AFI logo)
  display();
}
inline void Adafruit_PCD8544::spiWrite(uint8_t d) {
  if (isHardwareSPI()) {
    // Hardware SPI write.
    SPI.transfer(d);
  }
  else {
    // Software SPI write with bit banging.
    for(uint8_t bit = 0x80; bit; bit >>= 1) {
      *clkport &= ~clkpinmask;
      if(d & bit) *mosiport |=  mosipinmask;
      else        *mosiport &= ~mosipinmask;
      *clkport |=  clkpinmask;
    }
  }
}
bool Adafruit_PCD8544::isHardwareSPI() {
  return (_din == -1 && _sclk == -1);
}
void Adafruit_PCD8544::command(uint8_t c) {
  digitalWrite(_dc, LOW);
  if (_cs > 0)
    digitalWrite(_cs, LOW);
  spiWrite(c);
  if (_cs > 0)
```

```
    digitalWrite(_cs, HIGH);
}
void Adafruit_PCD8544::data(uint8_t c) {
  digitalWrite(_dc, HIGH);
  if (_cs > 0)
    digitalWrite(_cs, LOW);
  spiWrite(c);
  if (_cs > 0)
    digitalWrite(_cs, HIGH);
}
void Adafruit_PCD8544::setContrast(uint8_t val) {
  if (val > 0x7f) {
    val = 0x7f;
  }
  command(PCD8544_FUNCTIONSET | PCD8544_EXTENDEDINSTRUCTION );
  command( PCD8544_SETVOP | val);
  command(PCD8544_FUNCTIONSET);

 }
void Adafruit_PCD8544::display(void) {
  uint8_t col, maxcol, p;

  for(p = 0; p < 6; p++) {
#ifdef enablePartialUpdate
    // check if this page is part of update
    if ( yUpdateMin >= ((p+1)*8) ) {
      continue;   // nope, skip it!
```

```
    }
    if (yUpdateMax < p*8) {
      break;
    }
#endif
    command(PCD8544_SETYADDR | p);
#ifdef enablePartialUpdate
    col = xUpdateMin;
    maxcol = xUpdateMax;
#else
    // start at the beginning of the row
    col = 0;
    maxcol = LCDWIDTH-1;
#endif
    command(PCD8544_SETXADDR | col);
    digitalWrite(_dc, HIGH);
    if (_cs > 0)
      digitalWrite(_cs, LOW);
    for(; col <= maxcol; col++) {
      spiWrite(pcd8544_buffer[(LCDWIDTH*p)+col]);
    }
    if (_cs > 0)
      digitalWrite(_cs, HIGH);
  }
  command(PCD8544_SETYADDR );  // no idea why this is necessary but it is to finish
the last byte?
#ifdef enablePartialUpdate
  xUpdateMin = LCDWIDTH - 1;
```

```
  xUpdateMax = 0;
  yUpdateMin = LCDHEIGHT-1;
  yUpdateMax = 0;
#endif
}
// clear everything
void Adafruit_PCD8544::clearDisplay(void) {
  memset(pcd8544_buffer, 0, LCDWIDTH*LCDHEIGHT/8);
  updateBoundingBox(0, 0, LCDWIDTH-1, LCDHEIGHT-1);
  cursor_y = cursor_x = 0;
}
/*
// this doesnt touch the buffer, just clears the display RAM - might be handy
void Adafruit_PCD8544::clearDisplay(void) {

  uint8_t p, c;

  for(p = 0; p < 8; p++) {
    st7565_command(CMD_SET_PAGE | p);
    for(c = 0; c < 129; c++) {
      //uart_putw_dec(c);
      //uart_putchar(' ');
      st7565_command(CMD_SET_COLUMN_LOWER | (c & 0xf));
      st7565_command(CMD_SET_COLUMN_UPPER | ((c >> 4) & 0xf));
      st7565_data(0x0);
    }
    }
```

```
        }

        */
```

מהאתר http://playground.arduino.cc/Code/PCD8544

```
#define PIN_SCE   7
#define PIN_RESET 6
#define PIN_DC    5
#define PIN_SDIN  4
#define PIN_SCLK  3

#define LCD_C     LOW
#define LCD_D     HIGH

#define LCD_X     84
#define LCD_Y     48

static const byte ASCII[][5] =
{
 {0x00, 0x00, 0x00, 0x00, 0x00} // 20
,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #
,{0x24, 0x2a, 0x7f, 0x2a, 0x12} // 24 $
,{0x23, 0x13, 0x08, 0x64, 0x62} // 25 %
,{0x36, 0x49, 0x55, 0x22, 0x50} // 26 &
,{0x00, 0x05, 0x03, 0x00, 0x00} // 27 '
,{0x00, 0x1c, 0x22, 0x41, 0x00} // 28 (
,{0x00, 0x41, 0x22, 0x1c, 0x00} // 29 )
,{0x14, 0x08, 0x3e, 0x08, 0x14} // 2a *
,{0x08, 0x08, 0x3e, 0x08, 0x08} // 2b +
,{0x00, 0x50, 0x30, 0x00, 0x00} // 2c ,
,{0x08, 0x08, 0x08, 0x08, 0x08} // 2d -
```

```
,{0x00, 0x60, 0x60, 0x00, 0x00} // 2e .
,{0x20, 0x10, 0x08, 0x04, 0x02} // 2f /
,{0x3e, 0x51, 0x49, 0x45, 0x3e} // 30 0
,{0x00, 0x42, 0x7f, 0x40, 0x00} // 31 1
,{0x42, 0x61, 0x51, 0x49, 0x46} // 32 2
,{0x21, 0x41, 0x45, 0x4b, 0x31} // 33 3
,{0x18, 0x14, 0x12, 0x7f, 0x10} // 34 4
,{0x27, 0x45, 0x45, 0x45, 0x39} // 35 5
,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 36 6
,{0x01, 0x71, 0x09, 0x05, 0x03} // 37 7
,{0x36, 0x49, 0x49, 0x49, 0x36} // 38 8
,{0x06, 0x49, 0x49, 0x29, 0x1e} // 39 9
,{0x00, 0x36, 0x36, 0x00, 0x00} // 3a :
,{0x00, 0x56, 0x36, 0x00, 0x00} // 3b ;
,{0x08, 0x14, 0x22, 0x41, 0x00} // 3c <
,{0x14, 0x14, 0x14, 0x14, 0x14} // 3d =
,{0x00, 0x41, 0x22, 0x14, 0x08} // 3e >
,{0x02, 0x01, 0x51, 0x09, 0x06} // 3f ?
,{0x32, 0x49, 0x79, 0x41, 0x3e} // 40 @
,{0x7e, 0x11, 0x11, 0x11, 0x7e} // 41 A
,{0x7f, 0x49, 0x49, 0x49, 0x36} // 42 B
,{0x3e, 0x41, 0x41, 0x41, 0x22} // 43 C
,{0x7f, 0x41, 0x41, 0x22, 0x1c} // 44 D
,{0x7f, 0x49, 0x49, 0x49, 0x41} // 45 E
,{0x7f, 0x09, 0x09, 0x09, 0x01} // 46 F
,{0x3e, 0x41, 0x49, 0x49, 0x7a} // 47 G
,{0x7f, 0x08, 0x08, 0x08, 0x7f} // 48 H
,{0x00, 0x41, 0x7f, 0x41, 0x00} // 49 I
,{0x20, 0x40, 0x41, 0x3f, 0x01} // 4a J
,{0x7f, 0x08, 0x14, 0x22, 0x41} // 4b K
,{0x7f, 0x40, 0x40, 0x40, 0x40} // 4c L
,{0x7f, 0x02, 0x0c, 0x02, 0x7f} // 4d M
,{0x7f, 0x04, 0x08, 0x10, 0x7f} // 4e N
,{0x3e, 0x41, 0x41, 0x41, 0x3e} // 4f O
,{0x7f, 0x09, 0x09, 0x09, 0x06} // 50 P
,{0x3e, 0x41, 0x51, 0x21, 0x5e} // 51 Q
,{0x7f, 0x09, 0x19, 0x29, 0x46} // 52 R
,{0x46, 0x49, 0x49, 0x49, 0x31} // 53 S
,{0x01, 0x01, 0x7f, 0x01, 0x01} // 54 T
,{0x3f, 0x40, 0x40, 0x40, 0x3f} // 55 U
,{0x1f, 0x20, 0x40, 0x20, 0x1f} // 56 V
,{0x3f, 0x40, 0x38, 0x40, 0x3f} // 57 W
,{0x63, 0x14, 0x08, 0x14, 0x63} // 58 X
,{0x07, 0x08, 0x70, 0x08, 0x07} // 59 Y
,{0x61, 0x51, 0x49, 0x45, 0x43} // 5a Z
,{0x00, 0x7f, 0x41, 0x41, 0x00} // 5b [
,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c ¥
,{0x00, 0x41, 0x41, 0x7f, 0x00} // 5d ]
,{0x04, 0x02, 0x01, 0x02, 0x04} // 5e ^
,{0x40, 0x40, 0x40, 0x40, 0x40} // 5f _
,{0x00, 0x01, 0x02, 0x04, 0x00} // 60 `
,{0x20, 0x54, 0x54, 0x54, 0x78} // 61 a
,{0x7f, 0x48, 0x44, 0x44, 0x38} // 62 b
,{0x38, 0x44, 0x44, 0x44, 0x20} // 63 c
,{0x38, 0x44, 0x44, 0x48, 0x7f} // 64 d
,{0x38, 0x54, 0x54, 0x54, 0x18} // 65 e
,{0x08, 0x7e, 0x09, 0x01, 0x02} // 66 f
,{0x0c, 0x52, 0x52, 0x52, 0x3e} // 67 g
,{0x7f, 0x08, 0x04, 0x04, 0x78} // 68 h
,{0x00, 0x44, 0x7d, 0x40, 0x00} // 69 i
```

```
,{0x20, 0x40, 0x44, 0x3d, 0x00} // 6a j
,{0x7f, 0x10, 0x28, 0x44, 0x00} // 6b k
,{0x00, 0x41, 0x7f, 0x40, 0x00} // 6c l
,{0x7c, 0x04, 0x18, 0x04, 0x78} // 6d m
,{0x7c, 0x08, 0x04, 0x04, 0x78} // 6e n
,{0x38, 0x44, 0x44, 0x44, 0x38} // 6f o
,{0x7c, 0x14, 0x14, 0x14, 0x08} // 70 p
,{0x08, 0x14, 0x14, 0x18, 0x7c} // 71 q
,{0x7c, 0x08, 0x04, 0x04, 0x08} // 72 r
,{0x48, 0x54, 0x54, 0x54, 0x20} // 73 s
,{0x04, 0x3f, 0x44, 0x40, 0x20} // 74 t
,{0x3c, 0x40, 0x40, 0x20, 0x7c} // 75 u
,{0x1c, 0x20, 0x40, 0x20, 0x1c} // 76 v
,{0x3c, 0x40, 0x30, 0x40, 0x3c} // 77 w
,{0x44, 0x28, 0x10, 0x28, 0x44} // 78 x
,{0x0c, 0x50, 0x50, 0x50, 0x3c} // 79 y
,{0x44, 0x64, 0x54, 0x4c, 0x44} // 7a z
,{0x00, 0x08, 0x36, 0x41, 0x00} // 7b {
,{0x00, 0x00, 0x7f, 0x00, 0x00} // 7c |
,{0x00, 0x41, 0x36, 0x08, 0x00} // 7d }
,{0x10, 0x08, 0x08, 0x10, 0x08} // 7e ←
,{0x78, 0x46, 0x41, 0x46, 0x78} // 7f →
};

void LcdCharacter(char character)
{
  LcdWrite(LCD_D, 0x00);
  for (int index = 0; index < 5; index++)
  {
    LcdWrite(LCD_D, ASCII[character - 0x20][index]);
  }
  LcdWrite(LCD_D, 0x00);
}

void LcdClear(void)
{
  for (int index = 0; index < LCD_X * LCD_Y / 8; index++)
  {
    LcdWrite(LCD_D, 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_C, 0x21 );  // LCD Extended Commands.
  LcdWrite(LCD_C, 0xB1 );  // Set LCD Vop (Contrast).
  LcdWrite(LCD_C, 0x04 );  // Set Temp coefficent. //0x04
  LcdWrite(LCD_C, 0x14 );  // LCD bias mode 1:48. //0x13
  LcdWrite(LCD_C, 0x20 );  // LCD Basic Commands
  LcdWrite(LCD_C, 0x0C );  // LCD in normal mode.
}

void LcdString(char *characters)
```

```
{
  while (*characters)
  {
    LcdCharacter(*characters++);
  }
}

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 setup(void)
{
  LcdInitialise();
  LcdClear();
  LcdString("Hello World!");
}

void loop(void)
{
}
```

A simple modified example of interfacing with the Nokia 3310 LCD that will print characters at an XY position on LCD and also will draw lines on LCD.

```
/*
This Code has extra features
including a XY positioning function on Display
and a Line Draw function on Nokia 3310 LCD
It is modded from the original
http://playground.arduino.cc/Code/PCD8544
*/
// Mods by Jim Park
// jim(^dOt^)buzz(^aT^)gmail(^dOt^)com
// hope it works for you
#define PIN_SCE   7  // LCD CS  .... Pin 3
#define PIN_RESET 6  // LCD RST .... Pin 1
#define PIN_DC    5  // LCD Dat/Com. Pin 5
#define PIN_SDIN  4  // LCD SPIDat . Pin 6
#define PIN_SCLK  3  // LCD SPIClk . Pin 4
                  // LCD Gnd .... Pin 2
                  // LCD Vcc .... Pin 8
                  // LCD Vlcd ... Pin 7

#define LCD_C     LOW
#define LCD_D     HIGH

#define LCD_X     84
#define LCD_Y     48
#define LCD_CMD   0

int a = 0;
```

```
static const byte ASCII[][5] =
{
 {0x00, 0x00, 0x00, 0x00, 0x00} // 20
,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #
,{0x24, 0x2a, 0x7f, 0x2a, 0x12} // 24 $
,{0x23, 0x13, 0x08, 0x64, 0x62} // 25 %
,{0x36, 0x49, 0x55, 0x22, 0x50} // 26 &
,{0x00, 0x05, 0x03, 0x00, 0x00} // 27 '
,{0x00, 0x1c, 0x22, 0x41, 0x00} // 28 (
,{0x00, 0x41, 0x22, 0x1c, 0x00} // 29 )
,{0x14, 0x08, 0x3e, 0x08, 0x14} // 2a *
,{0x08, 0x08, 0x3e, 0x08, 0x08} // 2b +
,{0x00, 0x50, 0x30, 0x00, 0x00} // 2c ,
,{0x08, 0x08, 0x08, 0x08, 0x08} // 2d -
,{0x00, 0x60, 0x60, 0x00, 0x00} // 2e .
,{0x20, 0x10, 0x08, 0x04, 0x02} // 2f /
,{0x3e, 0x51, 0x49, 0x45, 0x3e} // 30 0
,{0x00, 0x42, 0x7f, 0x40, 0x00} // 31 1
,{0x42, 0x61, 0x51, 0x49, 0x46} // 32 2
,{0x21, 0x41, 0x45, 0x4b, 0x31} // 33 3
,{0x18, 0x14, 0x12, 0x7f, 0x10} // 34 4
,{0x27, 0x45, 0x45, 0x45, 0x39} // 35 5
,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 36 6
,{0x01, 0x71, 0x09, 0x05, 0x03} // 37 7
,{0x36, 0x49, 0x49, 0x49, 0x36} // 38 8
,{0x06, 0x49, 0x49, 0x29, 0x1e} // 39 9
,{0x00, 0x36, 0x36, 0x00, 0x00} // 3a :
,{0x00, 0x56, 0x36, 0x00, 0x00} // 3b ;
,{0x08, 0x14, 0x22, 0x41, 0x00} // 3c <
,{0x14, 0x14, 0x14, 0x14, 0x14} // 3d =
,{0x00, 0x41, 0x22, 0x14, 0x08} // 3e >
,{0x02, 0x01, 0x51, 0x09, 0x06} // 3f ?
,{0x32, 0x49, 0x79, 0x41, 0x3e} // 40 @
,{0x7e, 0x11, 0x11, 0x11, 0x7e} // 41 A
,{0x7f, 0x49, 0x49, 0x49, 0x36} // 42 B
,{0x3e, 0x41, 0x41, 0x41, 0x22} // 43 C
,{0x7f, 0x41, 0x41, 0x22, 0x1c} // 44 D
,{0x7f, 0x49, 0x49, 0x49, 0x41} // 45 E
,{0x7f, 0x09, 0x09, 0x09, 0x01} // 46 F
,{0x3e, 0x41, 0x49, 0x49, 0x7a} // 47 G
,{0x7f, 0x08, 0x08, 0x08, 0x7f} // 48 H
,{0x00, 0x41, 0x7f, 0x41, 0x00} // 49 I
,{0x20, 0x40, 0x41, 0x3f, 0x01} // 4a J
,{0x7f, 0x08, 0x14, 0x22, 0x41} // 4b K
,{0x7f, 0x40, 0x40, 0x40, 0x40} // 4c L
,{0x7f, 0x02, 0x0c, 0x02, 0x7f} // 4d M
,{0x7f, 0x04, 0x08, 0x10, 0x7f} // 4e N
,{0x3e, 0x41, 0x41, 0x41, 0x3e} // 4f O
,{0x7f, 0x09, 0x09, 0x09, 0x06} // 50 P
,{0x3e, 0x41, 0x51, 0x21, 0x5e} // 51 Q
,{0x7f, 0x09, 0x19, 0x29, 0x46} // 52 R
,{0x46, 0x49, 0x49, 0x49, 0x31} // 53 S
,{0x01, 0x01, 0x7f, 0x01, 0x01} // 54 T
,{0x3f, 0x40, 0x40, 0x40, 0x3f} // 55 U
,{0x1f, 0x20, 0x40, 0x20, 0x1f} // 56 V
,{0x3f, 0x40, 0x38, 0x40, 0x3f} // 57 W
,{0x63, 0x14, 0x08, 0x14, 0x63} // 58 X
```

```
,{0x07, 0x08, 0x70, 0x08, 0x07} // 59 Y
,{0x61, 0x51, 0x49, 0x45, 0x43} // 5a Z
,{0x00, 0x7f, 0x41, 0x41, 0x00} // 5b [
,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c ¥
,{0x00, 0x41, 0x41, 0x7f, 0x00} // 5d ]
,{0x04, 0x02, 0x01, 0x02, 0x04} // 5e ^
,{0x40, 0x40, 0x40, 0x40, 0x40} // 5f _
,{0x00, 0x01, 0x02, 0x04, 0x00} // 60 `
,{0x20, 0x54, 0x54, 0x54, 0x78} // 61 a
,{0x7f, 0x48, 0x44, 0x44, 0x38} // 62 b
,{0x38, 0x44, 0x44, 0x44, 0x20} // 63 c
,{0x38, 0x44, 0x44, 0x48, 0x7f} // 64 d
,{0x38, 0x54, 0x54, 0x54, 0x18} // 65 e
,{0x08, 0x7e, 0x09, 0x01, 0x02} // 66 f
,{0x0c, 0x52, 0x52, 0x52, 0x3e} // 67 g
,{0x7f, 0x08, 0x04, 0x04, 0x78} // 68 h
,{0x00, 0x44, 0x7d, 0x40, 0x00} // 69 i
,{0x20, 0x40, 0x44, 0x3d, 0x00} // 6a j
,{0x7f, 0x10, 0x28, 0x44, 0x00} // 6b k
,{0x00, 0x41, 0x7f, 0x40, 0x00} // 6c l
,{0x7c, 0x04, 0x18, 0x04, 0x78} // 6d m
,{0x7c, 0x08, 0x04, 0x04, 0x78} // 6e n
,{0x38, 0x44, 0x44, 0x44, 0x38} // 6f o
,{0x7c, 0x14, 0x14, 0x14, 0x08} // 70 p
,{0x08, 0x14, 0x14, 0x18, 0x7c} // 71 q
,{0x7c, 0x08, 0x04, 0x04, 0x08} // 72 r
,{0x48, 0x54, 0x54, 0x54, 0x20} // 73 s
,{0x04, 0x3f, 0x44, 0x40, 0x20} // 74 t
,{0x3c, 0x40, 0x40, 0x20, 0x7c} // 75 u
,{0x1c, 0x20, 0x40, 0x20, 0x1c} // 76 v
,{0x3c, 0x40, 0x30, 0x40, 0x3c} // 77 w
,{0x44, 0x28, 0x10, 0x28, 0x44} // 78 x
,{0x0c, 0x50, 0x50, 0x50, 0x3c} // 79 y
,{0x44, 0x64, 0x54, 0x4c, 0x44} // 7a z
,{0x00, 0x08, 0x36, 0x41, 0x00} // 7b {
,{0x00, 0x00, 0x7f, 0x00, 0x00} // 7c |
,{0x00, 0x41, 0x36, 0x08, 0x00} // 7d }
,{0x10, 0x08, 0x08, 0x10, 0x08} // 7e ←
,{0x00, 0x06, 0x09, 0x09, 0x06} // 7f →
};



void LcdCharacter(char character)
{
  LcdWrite(LCD_D, 0x00);
  for (int index = 0; index < 5; index++)
  {
    LcdWrite(LCD_D, ASCII[character - 0x20][index]);
  }
  LcdWrite(LCD_D, 0x00);
}

void LcdClear(void)
{
  for (int index = 0; index < LCD_X * LCD_Y / 8; index++)
  {
    LcdWrite(LCD_D, 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);
 // delay(1);
  digitalWrite(PIN_RESET, HIGH);

  LcdWrite( LCD_CMD, 0x21 );  // LCD Extended Commands.
  LcdWrite( LCD_CMD, 0xBf );  // Set LCD Vop (Contrast). //B1
  LcdWrite( LCD_CMD, 0x04 );  // Set Temp coefficent. //0x04
  LcdWrite( LCD_CMD, 0x14 );  // LCD bias mode 1:48. //0x13
  LcdWrite( LCD_CMD, 0x0C );  // LCD in normal mode. 0x0d for inverse
  LcdWrite(LCD_C, 0x20);
  LcdWrite(LCD_C, 0x0C);
}

void LcdString(char *characters)
{
  while (*characters)
  {
    LcdCharacter(*characters++);
  }
}

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);
}

// gotoXY routine to position cursor
// x - range: 0 to 84
// y - range: 0 to 5

void gotoXY(int x, int y)
{
  LcdWrite( 0, 0x80 | x);  // Column.
  LcdWrite( 0, 0x40 | y);  // Row.

}



void drawLine(void)
{
  unsigned char  j;
   for(j=0; j<84; j++) // top
                {
          gotoXY (j,0);
                   LcdWrite (1,0x01);
  }
```

```
  for(j=0; j<84; j++) //Bottom
                {
         gotoXY (j,5);
                  LcdWrite (1,0x80);
  }

  for(j=0; j<6; j++) // Right
                {
         gotoXY (83,j);
                  LcdWrite (1,0xff);
  }
                for(j=0; j<6; j++) // Left
                {
         gotoXY (0,j);
                  LcdWrite (1,0xff);
  }

}


void setup(void)
{

 LcdInitialise();
  LcdClear();

}

void loop(void)
{
  // Display some simple character animation
  //
  int a,b;
  char Str[15];
  // Draw a Box
  for(b=1000; b>0; b--){
  drawLine();
  for(a=0; a<=5 ; a++){
  gotoXY(4,1);
  // Put text in Box
  LcdString ("TestDisplay");
  gotoXY(24,3);
  LcdCharacter('H');
  LcdCharacter('E');
  LcdCharacter('L');
  LcdCharacter('L');
  LcdCharacter('O');
  LcdCharacter(' ');
  LcdCharacter('=');
  // Draw + at this position
  gotoXY(10,3);
  LcdCharacter('=');
  delay(500);
  gotoXY(24,3);
  LcdCharacter('h');
  LcdCharacter('e');
  LcdCharacter('l');
  LcdCharacter('l');
  LcdCharacter('o');
  LcdCharacter(' ');
```

```
  LcdCharacter('-');
  // Draw - at this position
  gotoXY(10,3);
  LcdCharacter('-');
  delay(500);
  }
  }
}
```

Another example which takes a bitmap via the serial port.

```
#define SER_BAUD  9600

#define PIN_SCE   7
#define PIN_RESET 6
#define PIN_DC    5
#define PIN_SDIN  4
#define PIN_SCLK  3

#define LCD_C     LOW
#define LCD_D     HIGH

void LcdClear(void)
{
  for (int index = 0; index < 84 * 48 / 8; index++)
  {
    LcdWrite(LCD_D, 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_C, 0x22);
  LcdWrite(LCD_C, 0x0C);
  LcdClear();
}

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 SerialInitialise(void) {
  Serial.begin(SER_BAUD);
}

void SerialRead(void) {
  if (Serial.available())
```

```
  {
    while (Serial.available())
    {
      LcdWrite(LCD_D, Serial.read());
    }
  }
}

void setup(void)
{
 LcdInitialise();
 SerialInitialise();
}

void loop(void)
{
 SerialRead();
}
```

And here's some sample VB.NET code to send bitmaps (loaded from file and generated on the fly) to the Arduino's serial port.

```
Serial_Write(New Bitmap("84x48.bmp"))
Serial_Write(Format(Now(), "HHmm"))

Private Sub Serial_Write(ByVal theString As String)
    Dim theBitmap As Bitmap = New Bitmap(84, 48)
    Dim theFont As Font = New Font("Courier", "24", FontStyle.Bold, GraphicsUnit.Pixel)
    Dim theGraphics As Graphics = Graphics.FromImage(theBitmap)
    theGraphics.TextRenderingHint = Drawing.Text.TextRenderingHint.ClearTypeGridFit
    theGraphics.FillRectangle(Brushes.White, 0, 0, theBitmap.Width, theBitmap.Height)
    theGraphics.DrawString(theString, theFont, Brushes.Black, ((theBitmap.Width -
theGraphics.MeasureString(theString, theFont).Width) / 2), ((theBitmap.Height -
theGraphics.MeasureString(theString, theFont).Height) / 2))
    Serial_Write(theBitmap)
End Sub

Private Sub Serial_Write(ByVal theBitMap As Bitmap)
    Dim theByteArray() As Byte = New Byte() {}
    For theWidth As Integer = 0 To 83
        For theHeight As Integer = 0 To 5
            ReDim Preserve theByteArray(theByteArray.GetUpperBound(0) + 1)
            For theBit As Integer = 0 To 7
                If theBitMap.GetPixel(theWidth, (theHeight * 8) + theBit).R Then
                    theByteArray(theByteArray.GetUpperBound(0)) =
theByteArray(theByteArray.GetUpperBound(0)) And Not (2 ^ theBit)
                Else
                    theByteArray(theByteArray.GetUpperBound(0)) =
theByteArray(theByteArray.GetUpperBound(0)) Or (2 ^ theBit)
                End If
            Next
        Next
    Next
    SerialPort.Open()
    SerialPort.Write(theByteArray, 0, theByteArray.Length)
    SerialPort.Close()
End Sub
```

Here is a Java version similar to the VB.net code above except that output goes to standard out (allows copy/paste of hex values into your sketch)

```
import java.awt.image.BufferedImage;
import java.io.File;

import javax.imageio.ImageIO;

public class BitmapToLCD {
	public static final int WIDTH = 84;
	public static final int HEIGHT = 48;

	public static void main(String[] args) {
		File f = new File(args[0]);

		try {
			// Read from a file
		BufferedImage image = ImageIO.read(f);


		// Get all the pixels
		int w = image.getWidth(null);
		int h = image.getHeight(null);
		int[] rgbs = new int[w*h];
		image.getRGB(0, 0, w, h, rgbs, 0, w);

		//iterate through each pixel (and reduce to binary)
		int row = 0;
		int col = 0;
		int bit = 0;
		byte[][] ba = new byte[HEIGHT/8][WIDTH];
		for (int i = 0; i < rgbs.length; i++){
			byte val = (byte)(rgbs[i] & 0x01);
			//invert the value
			val = (byte) (val == 1 ? 0:1);
				ba[row][col] |= val << bit;

			//next column
			col++;

			//next bit
			if (col >=WIDTH) {
				col = 0;
				bit++;
			}

			//next data row
			if (bit >=8){
			   bit = 0;
			  for (int x= 0; x < WIDTH; x++){
				String s = Integer.toHexString((byte)ba[row][x]);
				//Do some formatting
				if (s.length() > 2) {
					s = s.substring(s.length() - 2);
				}
				while (s.length() < 2){
					s = "0" + s;
```

```
                                        }
                                    System.out.print( "0x" + s + ",");
                                }
                                System.out.println("");
                                row++;
                            }

                        }
                            } catch (Exception e) {
                                    e.printStackTrace();
                            }

                }
}
```