This is not an official Arduino product

This is a third party effort and has not been endorsed by anyone affiliated with Arduino.

Contents

1. Physically connecting the Powertip PG320240WRFHE9 to your Arduino

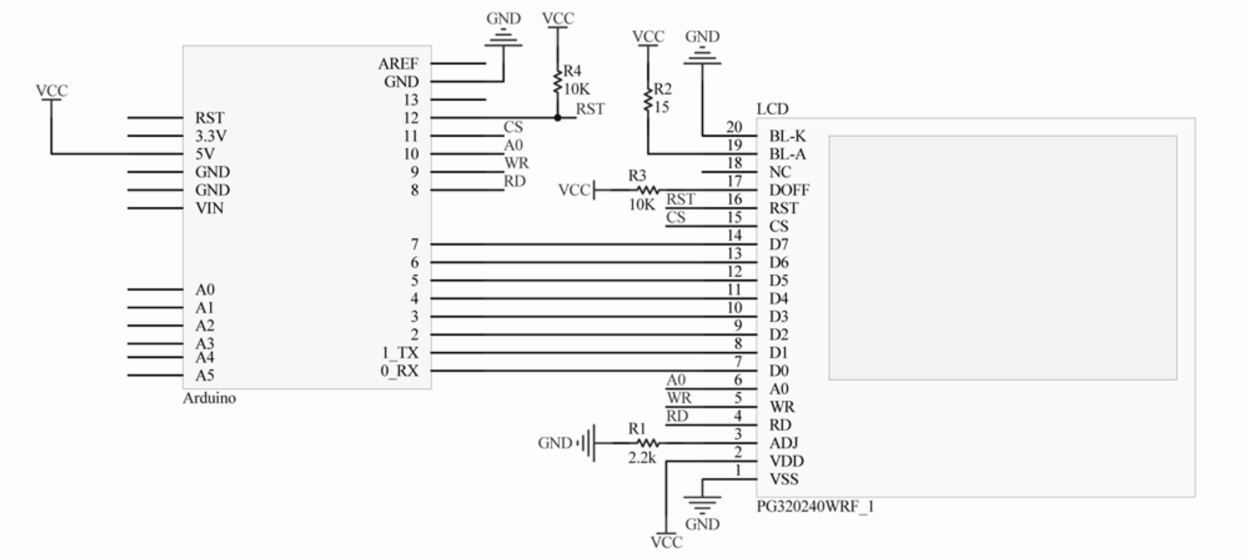
If possible, the Arduino should be connected according to the provided schematic (Figure A). The provided connections will provide configuration free usage for most Arduino boards. Some Arduino boards, such as the Mega2560, will require custom configuration. This is determined by how the ports on the Atmel AVR microcontroller are mapped to the digital pins as broken out on your Arduino board. Most Arduino boards, including the Uno, Duemilanove, and Nano, have port D mapped to digital pins 0 through 7; these boards will work without any configuration changes.

For more information about AVR ports as they relate to the Arduino visit: http://arduino.cc/en/Reference/PortManipulation

To determine if your Arduino will work with the default configuration, view the schematic for your board. It is available here: http://arduino.cc/en/Main/Boards

If your board will not support the default configuration, refer to the section of this manual covering software setup with custom pin settings.

Figure A



Note: The value of R1 may need to be adjusted to obtain optimal contrast. However, it should be a good starting place for testing the display.

2. Importing the software library

Refer to “Contributed Libraries”: http://arduino.cc/en/Reference/Libraries

3. Setting up the Software Library

3a. Setting up the software library with the default connections

If you’ve used the default connections, there is not configuration required. All you need to do to start displaying information is create an S1D13700 object and call the initLCD function. See the example sketch for more information.

```
/*Example initialization with default connections */

#include <S1D13700.h>

/*create our S1D13700 LCD object and name it
 glcd. */
S1D13700 glcd;

void setup()
{
  /*Call the setup routine */
  glcd.initLCD();

  /*Create a string variable */
  char buf[] = "Hello World!";

  /*Clear the screen */
  glcd.clearText();
  glcd.clearGraphic();

  /*specify the column and row address
   where the text should be output */
  glcd.textGoTo(10,1);

  /*Write our string variable to the screen */
  glcd.writeText(buf);
}
```

3b. Setting up the software library with custom control pin connections

If you only need to change the configuration of the control pins (pins other than D0-D7), the procedure is very similar to section 3a. The difference is that the pin connections must be specified after you create the S1D13700 object and before you call the initLCD function.

```
/*Example initialization with custom control pins */

#include <S1D13700.h>

/*create our S1D13700 LCD object and name it
 glcd. */
S1D13700 glcd;

void setup()
{
  /*Specify control pin connections*/
  glcd.pins.rd = 10;
  glcd.pins.wr = 8;
  glcd.pins.a0 = 13;
  glcd.pins.cs = 11;
  glcd.pins.rst = 12;

  /*Call the setup routine */
  glcd.initLCD();

  /*Create a string variable */
  char buf[] = "Hello World!";

  /*Clear the screen */
  glcd.clearText();
  glcd.clearGraphic();

  /*specify the column and row address
   where the text should be output */
  glcd.textGoTo(10,1);

  /*Write our string variable to the screen */
  glcd.writeText(buf);
}
```

3c. Setting up the software library with fully custom connections

If you are using a board which does not have AVR PORTD or digital pins 0 through 7 are unavailable to the LCD you have two options. Option 1 is to modify the S1D13700.h library by changing the port specified. Option 1 is superior from a performance standpoint. Option 2 is to modify the S1D13700.h library by un-commenting the S1D13700_CUSTOM_DATA_PINS definition, then specifying individual pin settings for each data pin. Option 2 is the most flexible option but it will result in significantly degraded performance.

Option 1: Changing the Data Port

Example: Bob has an Arduino Mega2560. After reviewing the schematic for the Arduino Mega2560, bob determines that port D is not entirely broken out and is therefore not useable as a data port. However, Bob notices that port C, although in reverse order, is entirely available as digital pins 30 through 37. Bob connects LCD data pin D0 to Arduino pin 37, then continues, finishing with LCD pin D7 being connected to Arduino pin 30. In order to let the software library know about his physical change, Bob opens the S1D13700.h library file. He finds the "FIXED_PORT" definitions and makes the following changes.

```
//#define S1D13700_CUSTOM_DATA_PINS

#define FIXED_DIR DDRD
#define FIXED_PORT PORTD
#define FIXED_PIN PIND
```

Is changed to:

```
//#define S1D13700_CUSTOM_DATA_PINS

#define FIXED_DIR DDRC
#define FIXED_PORT PORTC
#define FIXED_PIN PINC
```

Bob then proceeds successfully according to the set-up instructions in section 3b of this manual.

Option 2: Specifying custom pins

Example: Bob has an Arduino Duemilanove. Bob would love to make life easy on himself and use the default connections but he really needs the RX and TX pins for communication. Bob decides that he is willing to occasionally reset his Arduino after start up if the display does not properly initialize. Bob then foregoes use of the RST pin and simply connects the LCD RST pin to V+ through a pull up resistor. Bob also decides he does not need to write to the screen very quickly. Bob will be primarily using text and he is comfortable with graphics being drawn slowly. Since Arduino pins 0 and 1 are occupied, Bob connects the LCD data port to pins 2 through 9. He then connects the remaining LCD control pins to Arduino pins 10 through 13. In order to let the software library know that he will be using specific pins, Bob opens the S1D13700.h library file and uncomments this line:

```
#define S1D13700_CUSTOM_DATA_PINS
```

After saving his change to S1D13700.h, bob initializes the display with the following code:

```
/*Example initialization with custom data and control
pins */

#include <S1D13700.h>

/*create our S1D13700 LCD object and name it
 glcd. */
S1D13700 glcd;

void setup()
{
  /*Specify control pin connections*/
  glcd.pins.d0 = 2;
  glcd.pins.d1 = 3;
  glcd.pins.d2 = 4;
  glcd.pins.d3 = 5;
  glcd.pins.d4 = 6;
  glcd.pins.d5 = 7;
  glcd.pins.d6 = 8;
  glcd.pins.d7 = 9;
  glcd.pins.rd = 10;
  glcd.pins.wr = 11;
  glcd.pins.a0 = 12;
  glcd.pins.cs = 13;

  /*Call the setup routine */
  glcd.initLCD();

  /*Create a string variable */
  char buf[] = "Hello World!";

  /*Clear the screen */
  glcd.clearText();
  glcd.clearGraphic();

  /*specify the column and row address
   where the text should be output */
  glcd.textGoTo(10,1);

  /*Write our string variable to the screen */
  glcd.writeText(buf);
}
```

4. Library Function Reference

initLCD()
Description: Initilalizes the LCD.
Arguments: void
Remarks: This function must be called before any other LCD functions.
Return Value: void

writeText(char * text)
Description: Display a string from memory on the LCD.
Arguments:
text – the string to display
Remarks: Set the desired location of the text with textGoTo prior to calling this function.
Return Value: void

textGoTo(unsigned char x, unsigned char y)
Description: Moves the text cursor to the specified location.
Arguments:
x – The column to move the cursor to.
y – The row to move the cursor to.
Remarks: Rows and columns are not the same as pixels. A 320x240 screen will have 40 text rows and 30 text columns.
Return Value: void

clearText ()
Description: Clears the text layer.
Arguments: void
Remarks: This function will not affect the graphics layer.
Return Value: void

clearGraphic ()
Description: Clears the graphics layer.
Arguments: void
Remarks: This function will not affect the text layer.
Return Value: void

setPixel(unsigned int x,unsigned int y, unsigned char state)
Description: Clears or sets a single pixel.
Arguments:
x – The x location of the pixel.
y – The y location of the pixel.
state – The desired state of the pixel. This should be 0 for off or 1 for on.
Remarks: Pixel values start at 0. This function does not employ bounds checking.
Return Value: void

drawBox(int x0, int y0, int x1, int y1)
Description: Draws a rectangle.
Arguments:
x0 – The x location of the upper left corner.
y0 – The y location of the upper left corner.
x1 – The x location of the lower right corner.
y1 – The y location of the lower right corner.
Remarks: Pixel values start at 0. This function does not employ bounds checking.
Return Value: void

drawCircle(int x0, int y0, int radius)
Description: Draws a circle
Arguments:
x – The x location of the center.
y – The y location of the center.
radius – The radius of the circle in pixels.
Remarks: Pixel values start at 0. This function does not employ bounds checking.
Return Value: void

drawLine(int x0, int y0, int x1, int y1)
Description: Draws a line.
Arguments:
x0 – The x location of the origin.
y0 – The y location of the origin.
x1 – The x location of the destination.
y1 – The y location of the destination.
Remarks: Pixel values start at 0. This function does not employ bounds checking.
Return Value: void