**Setting Up Papilio Pro with 32x32 LED Smart Matrix**

**Initial Setup:**

First you’ll need to download Papilo design lab. Design lab allows you to edit code and deploy circuits to your FPGA module. The Design lab installation files can be found here:

<http://forum.gadgetfactory.net/index.php?/files/file/236-papilio-designlab-ide/>

To edit circuit designs you will also need to install the Xilinx ISE WebPack. You will need to register before you can download your files. Follow the installation prompts, if you do not want to choose a license deselect ‘Acquire or Manage a License Key’ when prompted.

<http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html>

You can find more detailed information about the Papilio Pro board layout here:

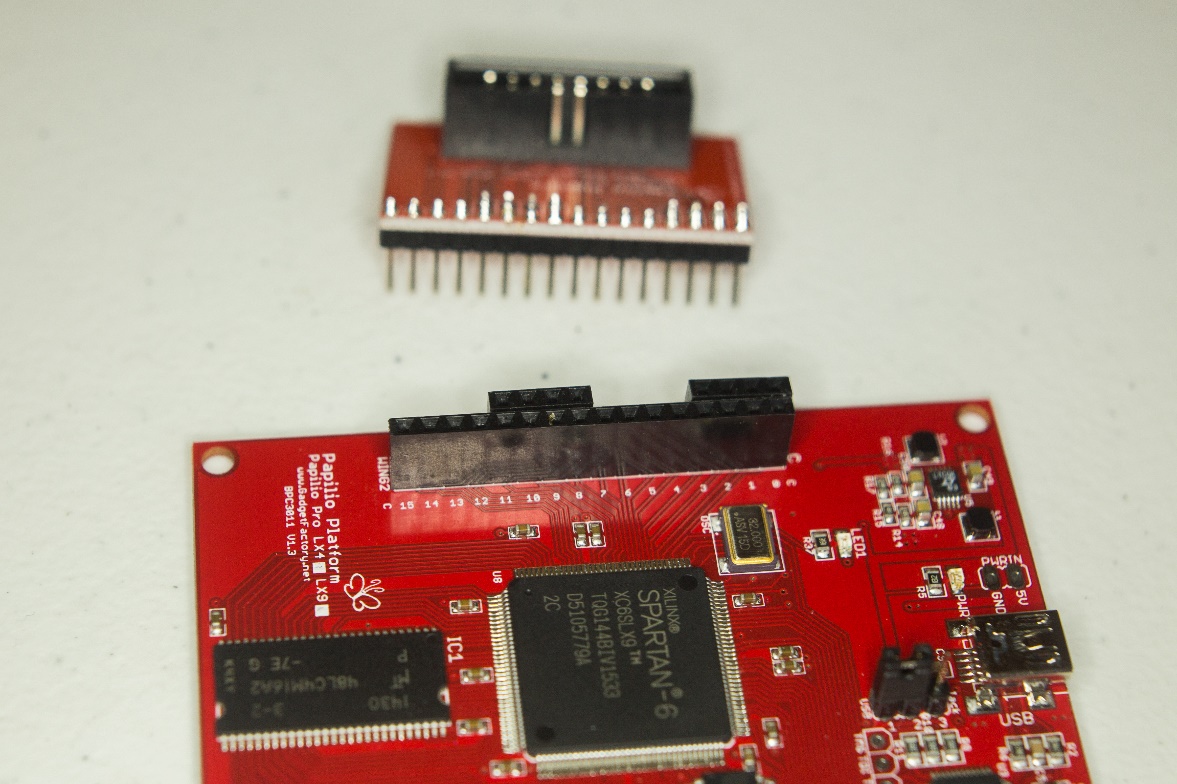
<http://papilio.cc/index.php?n=Papilio.PapilioPro>

Once you have Design lab you are ready to put your components together. You’ll need the following items to complete the build:

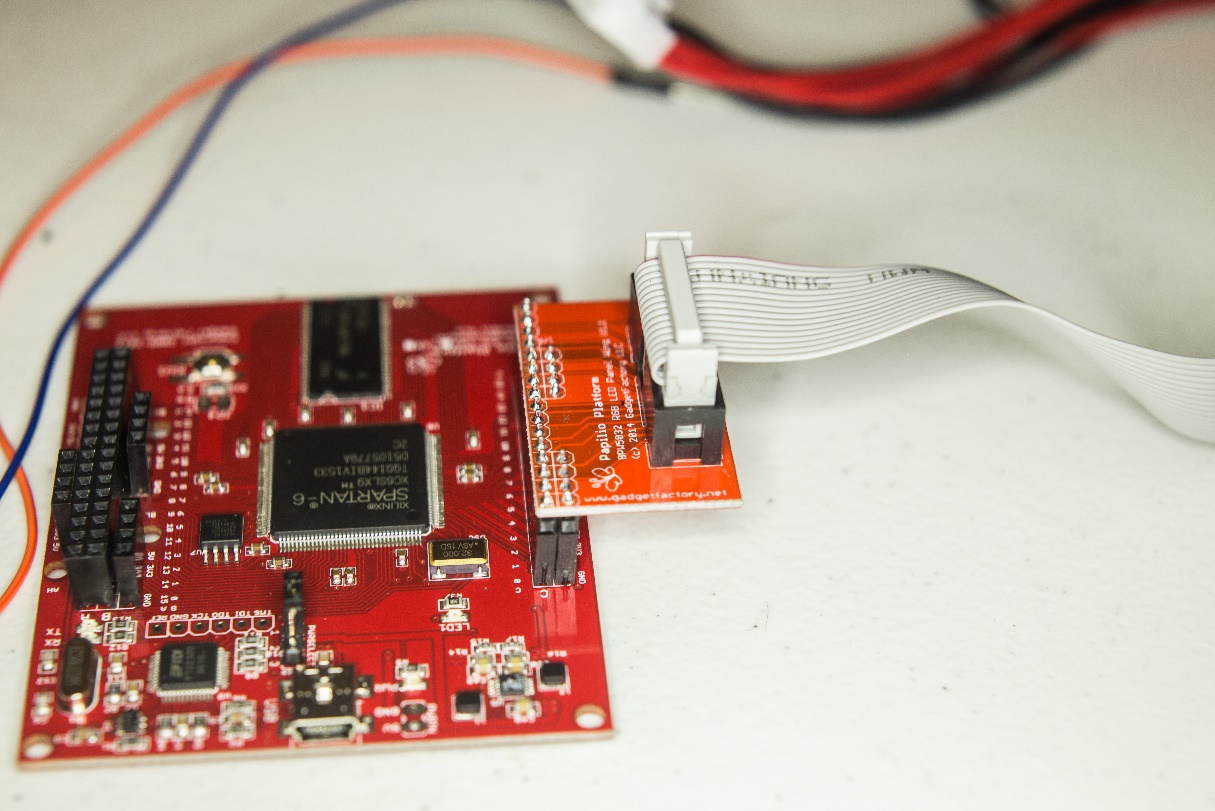
* 1 Papilio Pro module.
* 1 32x32 LED Smart Matrix.
* 1 Smart Matrix wing.
* 1 Smart Matrix data cable.
* 1 Smart Matrix power cable.
* 1 male-male USB A to USB MINI B connector.
* 2 male-male jumper wires.

**Hardware Setup:**

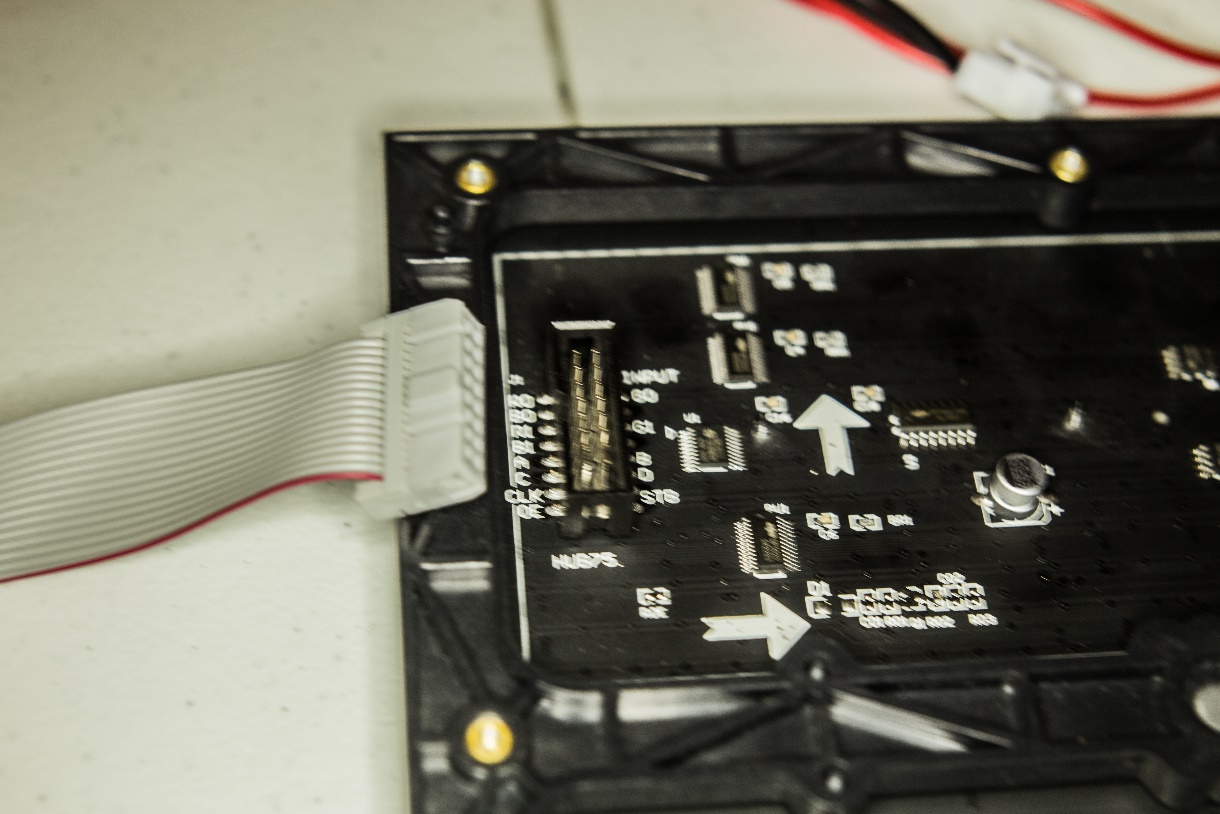
Plug the Smart Matrix shield module into the slot labeled WING2 of the Papilio Pro.



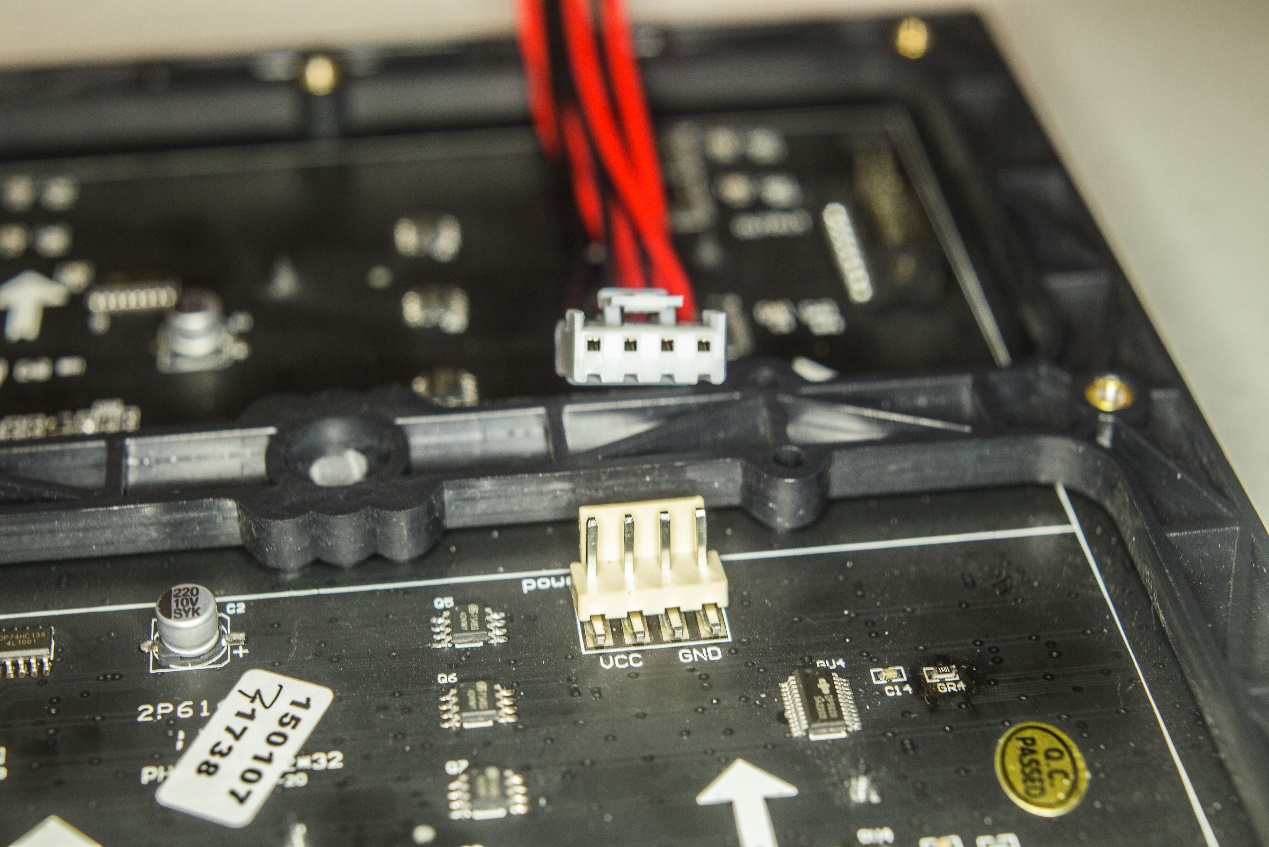
Next Connect your data cable to the Smart Matrix wing.



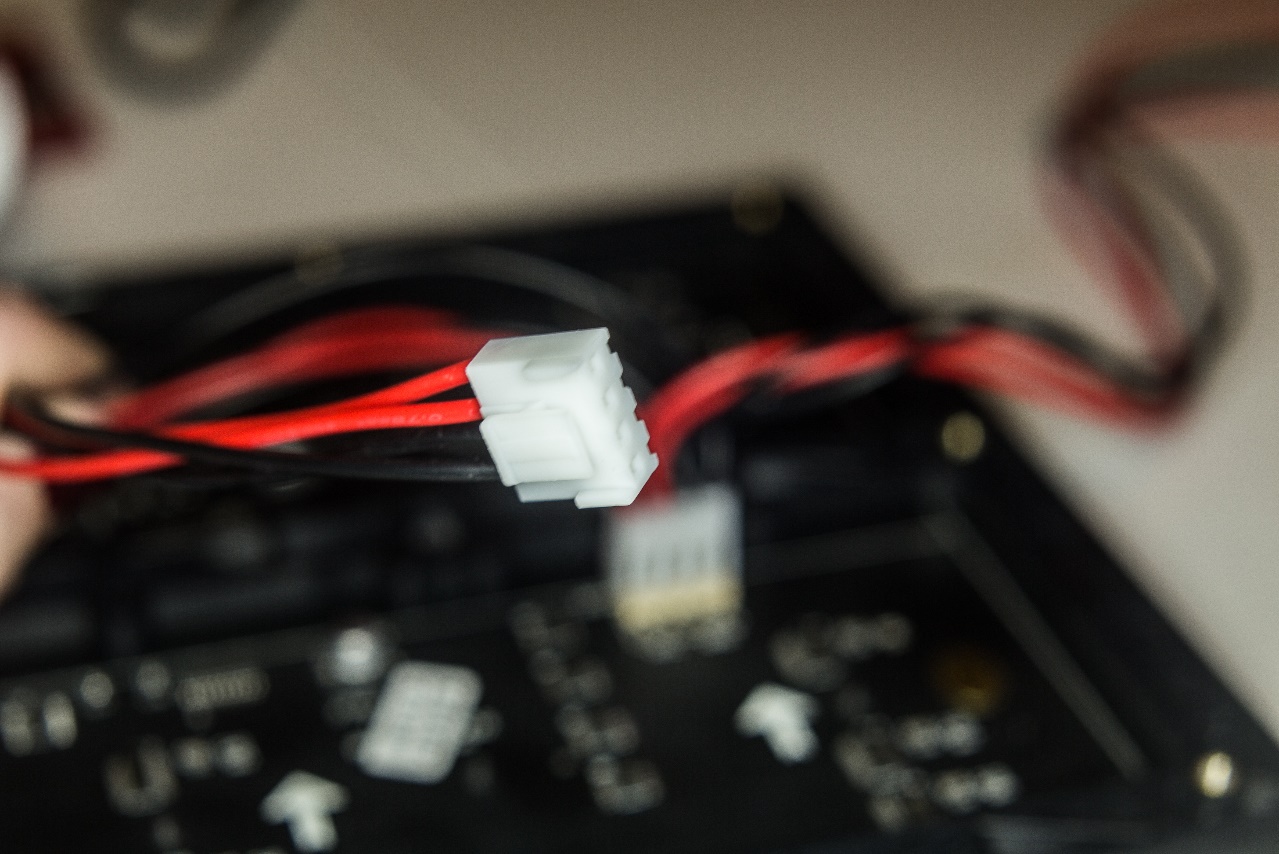
Turn over the LED matrix and locate the data port labeled Input.



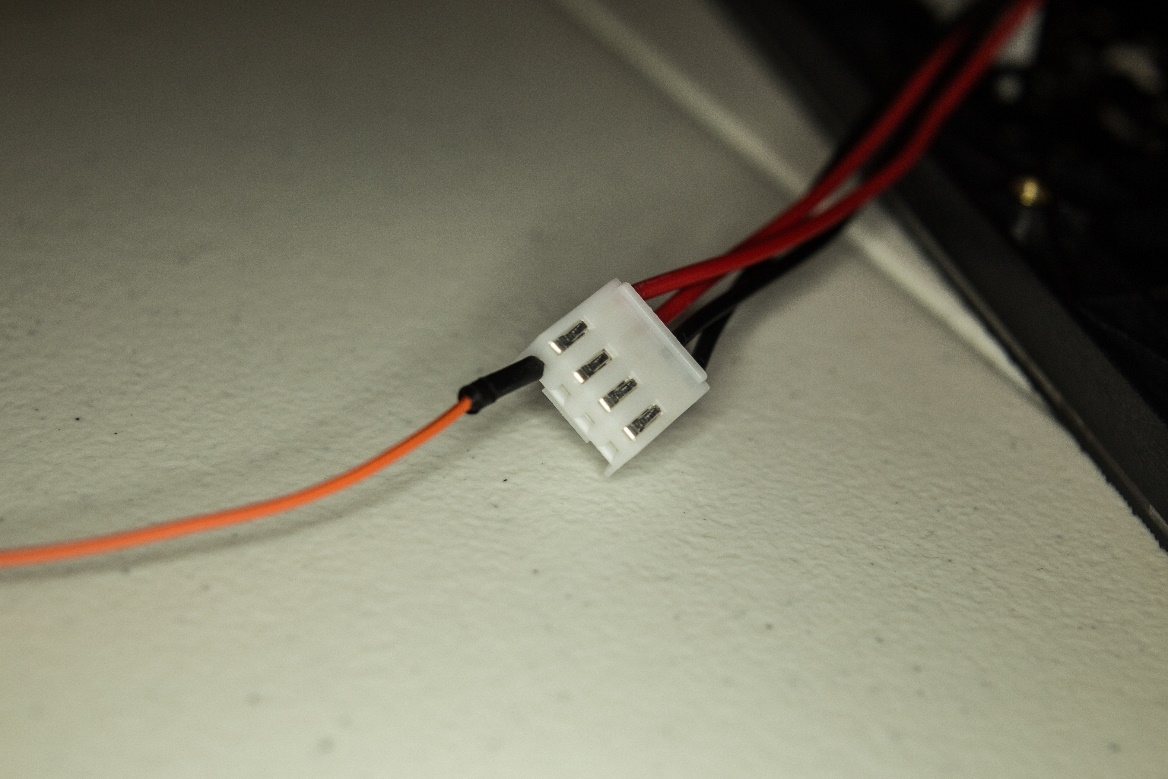
Next plug the power cable into the back of the LED matrix.



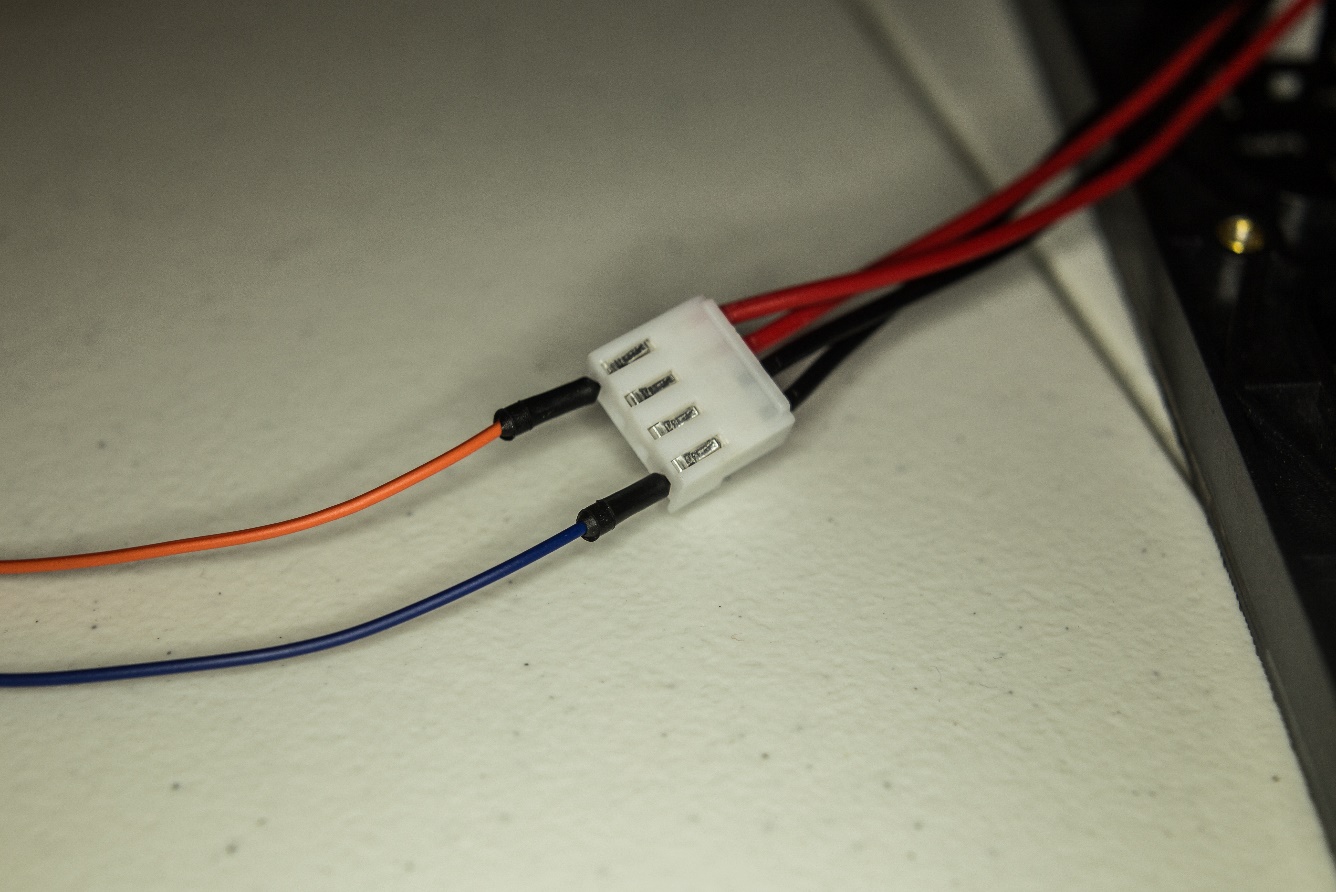
You’ll be left with a second plug and two leads.



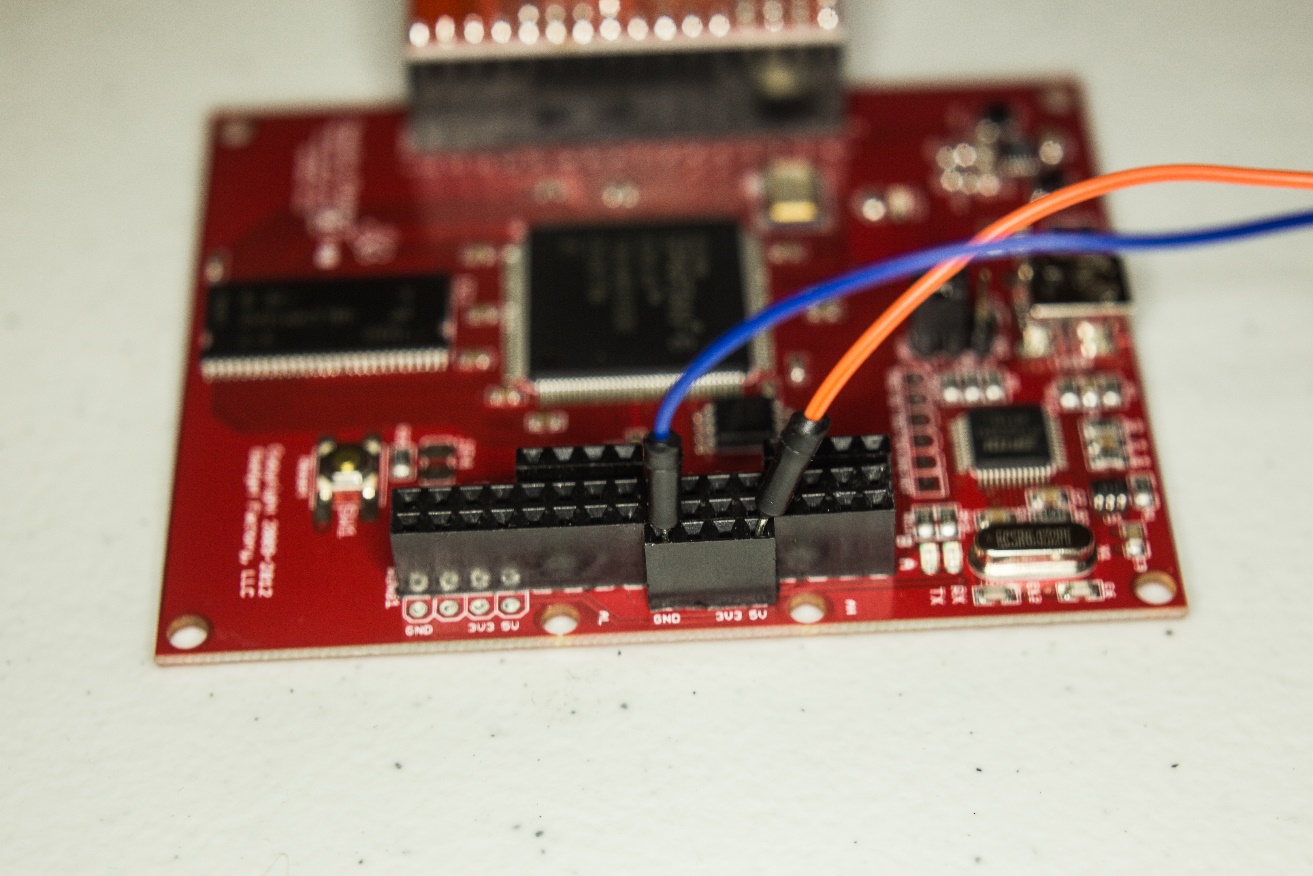
You will need to use the two jumpers to connect the second plug to the 5V power output on wing one of the Papilio Pro. Connect a red jumper to the outer red wire of the plug, this is the power into for the matrix.



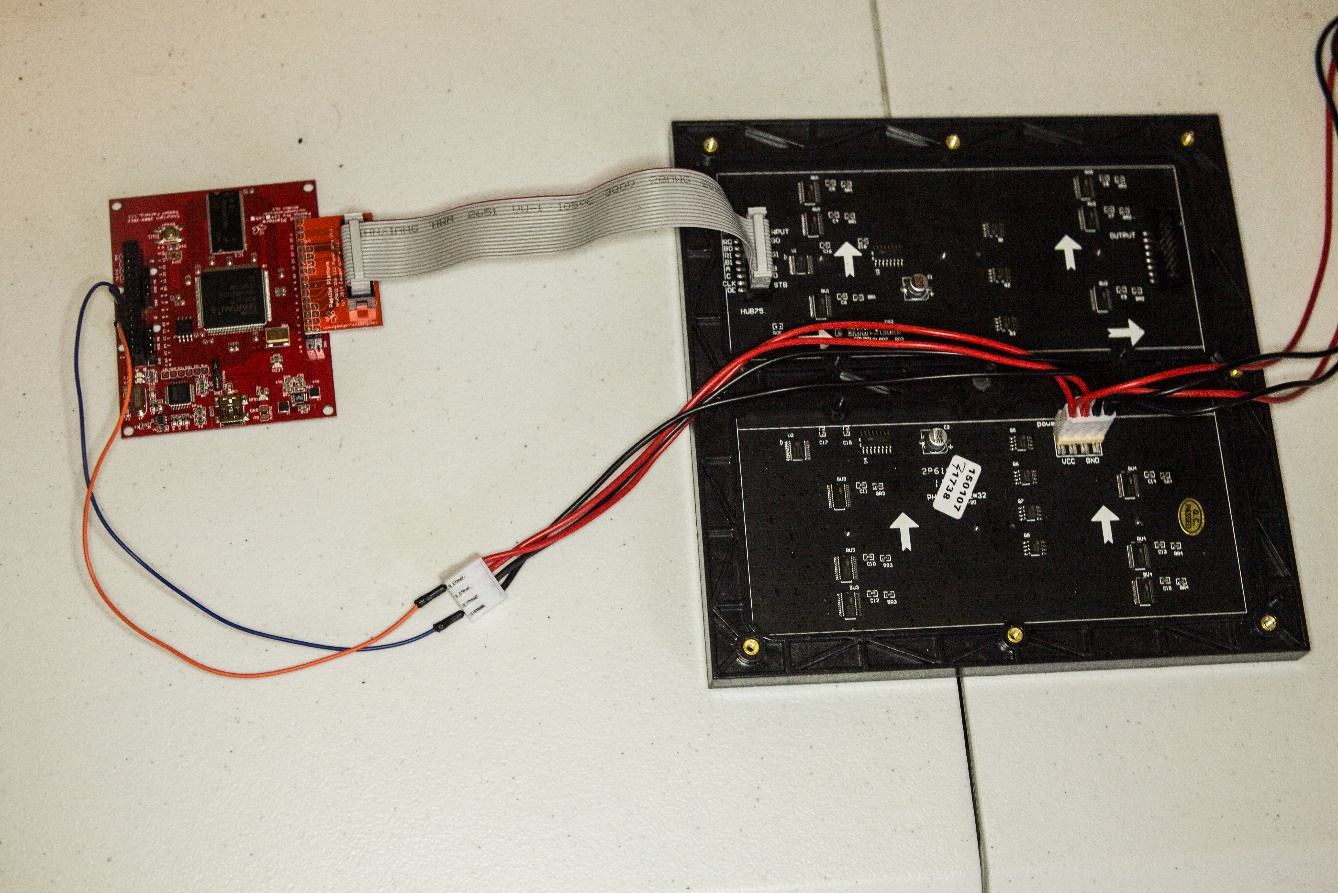
Next connect a black jumper to the outer black wire of the plug, this is the ground.



Plug the red jumper into the socket labeled 5V and the black jumper into the socket labeled GND.

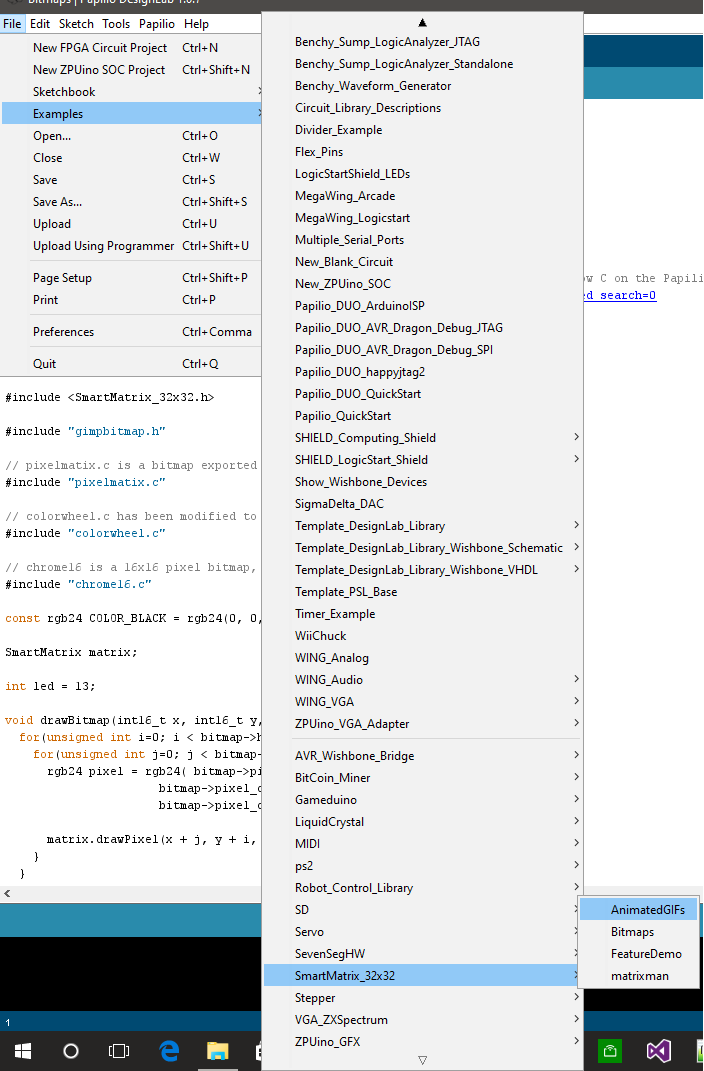


Your finished build should look like this. You’ll have two exposed leads, make sure these do not touch or you can short your device. You can use electrical tape to prevent accidental shorting.

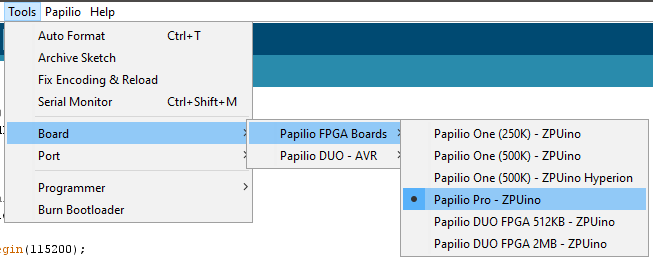


**FPGA Setup:**

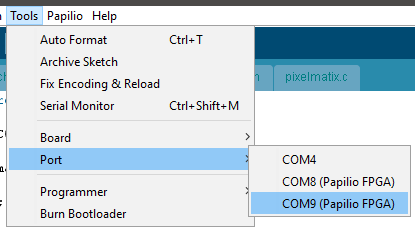
Now you should be ready to setup your FPGA. Plug your USB cable into a port on your PC and into the Papilio. You may see lights flash on the LED matrix but you still need to program your FPGA using design lab. There are several example programs for the LED matrix in design lab already. First load a sample using the File Menu > Examples > Smart Matrix\_32x32 menu option.



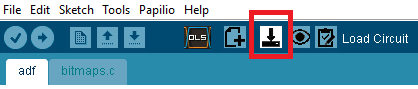
Now you’ll need to select your board, go to Tools > Board > Papilio FPGA Boards > Papilio Pro – Zpuino.



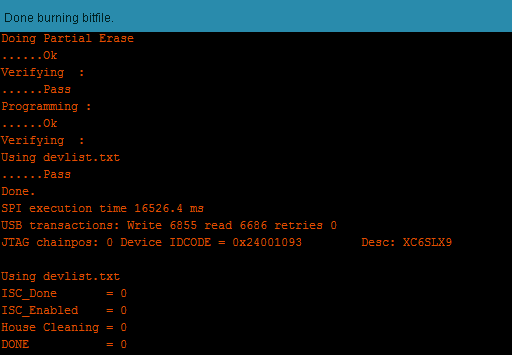
You then need to select your COM port from Tools > Port. There may be several listed, unplug you device and plug it in to see the list change. It’s usually the highest port but yours may differ from the sample.



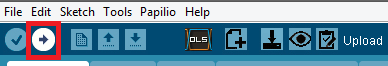
Now you’re ready to upload your circuit to the FPGA. Click the Load Circuit button in the toolbar.

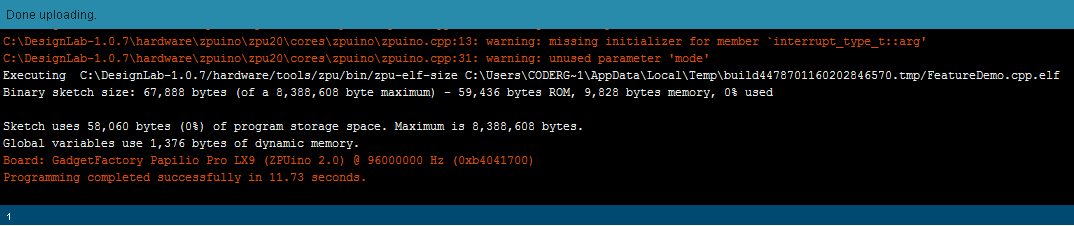


You should see output similar to the following in the console.

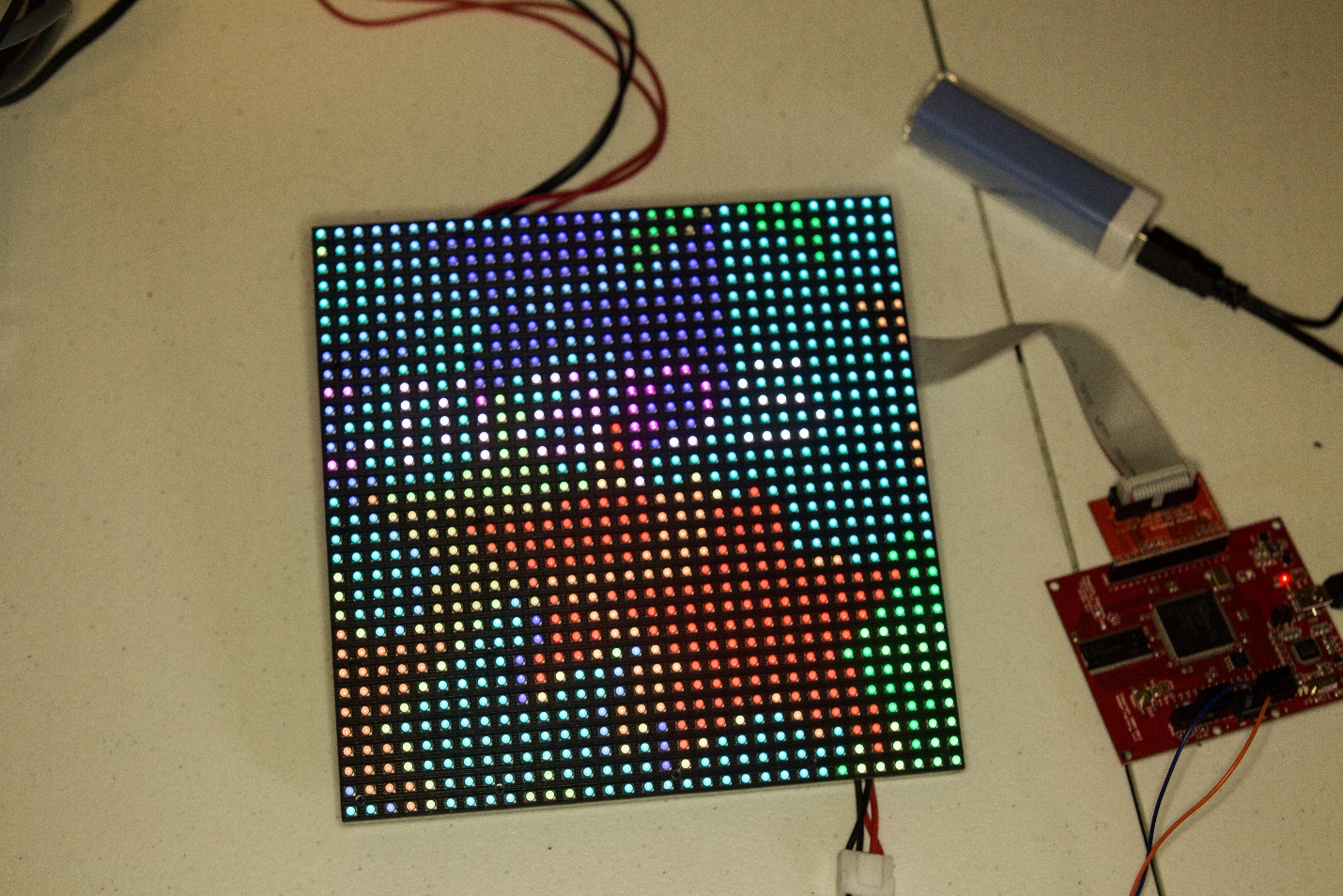


Now you’re ready to run the sample code. Click the Upload button in the toolbar.



You should see output similar to the following: 

Once it completes you should see the demo running on your LED matrix.



**Development Setup:**

Now that you have your board working, you can build and deploy new firmware for your SOC or you can design new circuits. You can either start from an existing example or start from scratch. To edit circuits you will need to be familiar with an HDL language, either VHDL or Verilog are available in the Xilinx ISE circuit editor.

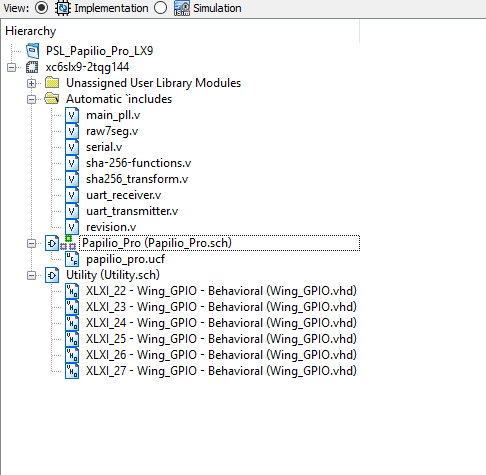
To start your own project you will need a new circuit project or a new SOC project depending on your implementation. In Design Lab choose File > New FPGA Circuit Project to create a new blank circuit.



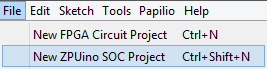
By default your new circuit project file is Read-Only you will need to save your file using File > Save As, or you will have the option to save your project when you move on to the next step. To edit your circuit click the Edit Circuit button in the toolbar. Make sure you have selected your board from the Tools > Board menu. This will launch the Xilinx ISE environment with an empty circuit project.



You should have a project similar to the following image. Your new project is pre-configured for the board you selected.



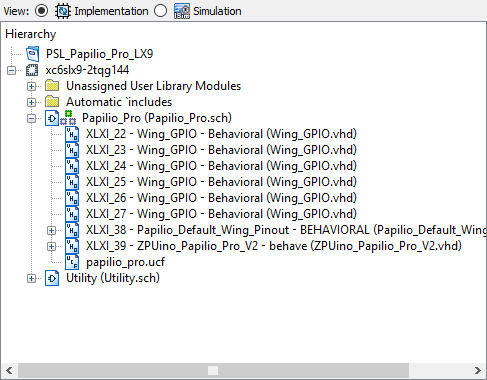
Alternatively, you can create an SOC project that will give you a built in soft processor preconfigured in your new circuit design. The steps for setting this up are similar but instead of choose Circuit Project from the main menu choose File > New ZPUino SOC Project.

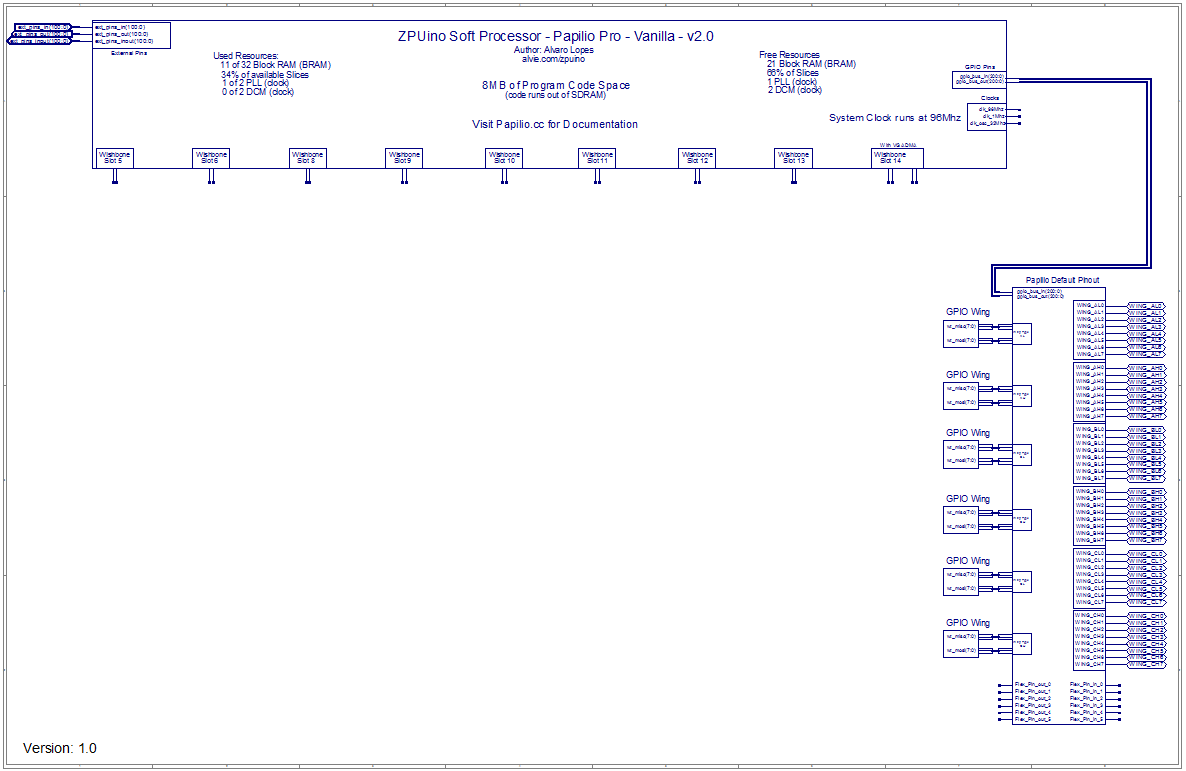


Make sure you have selected your board from the Tools > Board menu. This will launch the Xilinx ISE environment with the default SOC project.



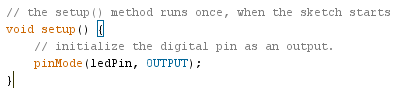
You should have a project similar to the following, depending on your board:



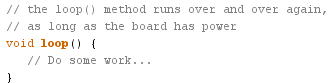
If you double click on the main circuit node of your project, Papilio\_Pro (Papilio\_Pro.sch) in this example, you can view the default SOC diagram created in your project.

Now you are ready to start adding your own modules, you have a few options, you can start with one of the sample projects included in Design Lab, or you can write your own circuits. You can also use Modules included in your Xilinx ISE project by default. They are located under the Unassigned User Library Modules node in the Design Hierarchy. There are also several open source resources available online, one of which is <http://www.opencores.org>.

Once you have a SOC project setup you will need some code to run on it. The Design Lab software allows you to create firmware that can run on your SOC, it uses a language called Sketch, which is similar to C/C++. In an SOC project there are two main functions, setup() and loop(), that you’ll need to implement in your Design Lab project. The setup() function is called before loop() starts processing, you can initialize resources or setup your pins for input or output here.



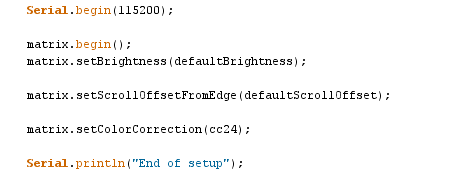
The loop() function is your main processing loop and is called continuously while your program is running.



To use the Smart Matrix display you need to include the Smart Matrix header file, SmartMatrix\_32x32.h, and create an instance of the SmartMatrix class.



In setup() you need to set your default smart matrix configuration.



|  |
| --- |
| There are several configuration options, below is a partial list of useful configuration function. |
| void setRotation(rotationDegrees rotation); // Set screen rotation 0,90,180, or 270 degrees |
| void setBrightness(uint8\_t brightness); // Sets brightness of pixels. |
| void setBackgroundBrightness(uint8\_t brightness); // Sets Background brightness. |
| void setColorCorrection(colorCorrectionModes mode); // Lets you set color correction modes. |
| void setFont(fontChoices newFont); // Sets current font. |
|  |

You can also query the vertical and horizontal extents of your screen in pixels using the matrix.getScreenWidth() and matrix.getScreenHeight(). If you wish to fill your screen with a single color before your rendering you can use the fillScreen() function.

void fillScreen(const rgb24& color);

Now that you have everything setup you can render pixels, strings, or other primitives using the built in drawing functions of the SmartMatrix library. These are the pixel and line drawing functions.

|  |
| --- |
| void drawPixel(int16\_t x, int16\_t y, const rgb24& color); |
| void drawLine(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, const rgb24& color); |
| void drawFastVLine(int16\_t x, int16\_t y0, int16\_t y1, const rgb24& color); |
| void drawFastHLine(int16\_t x0, int16\_t x1, int16\_t y, const rgb24& color); |

You can draw simple primitives like circles, triangles, and rectangles; you usually have the option of a drawing an outline or drawing a full primitive.

|  |
| --- |
| void fillCircle(int16\_t x0, int16\_t y0, uint16\_t radius, const rgb24& color); |
| void drawRectangle(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, const rgb24& color); |
| void fillRectangle(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, const rgb24& color); |

You can also draw characters and text to the screen. You specify the font for rendering characters or strings by calling the setFont() function before drawing your characters or strings.

|  |
| --- |
| void drawChar(int16\_t x, int16\_t y, const rgb24& charColor, char character); |
| void drawString(int16\_t x, int16\_t y, const rgb24& charColor, const char text[]); |
| void drawString(int16\_t x, int16\_t y, const rgb24& charColor, const rgb24& backColor, const char text[]); |

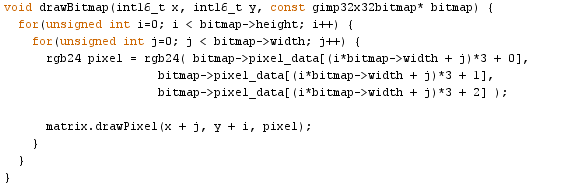
You can also control text scrolling through several functions; these allow you to set scroll text, color, font and speed.

|  |
| --- |
| void scrollText(const char inputtext[], int numScrolls); |
| void setScrollMode(ScrollMode mode); |
| void setScrollSpeed(unsigned char pixels\_per\_second); |
| void setScrollFont(fontChoices newFont); |
| void setScrollColor(const rgb24 & newColor); |

Finally, you can draw a full bitmap using the drawMonoBitmap. This will draw a monochromatic bitmap to your SmartMatrix.

void drawMonoBitmap(int16\_t x, int16\_t y, uint8\_t width, uint8\_t height, const rgb24& bitmapColor, const uint8\_t \*bitmap);

In order to draw a full color bitmap you need to implement a custom drawing function like the following.



After rendering your primitives you’ll want to swap your buffer using the swapBuffer() function.

void swapBuffers(bool copy = true);

For more advanced rendering, you can draw directly to the back buffer or create your own buffer and then render that using the following buffer management functions.

|  |
| --- |
| rgb24 \*backBuffer(void); |
| void setBackBuffer(rgb24 \*newBuffer); |
| rgb24 \*getRealBackBuffer(void); |

You can refer the examples provided in Design Lab for more information or you can find source code for the SmartMatrix library here:

<https://github.com/GadgetFactory/DesignLab_Examples/tree/master/libraries/SmartMatrix_32x32>

That should be enough to get started, Enjoy!