

# **TL25: KEY DOCUMENT**

## Validation

# Changelog

Date/Time	Updater	Description
<b>2018-10-14T14:40:56Z</b>	Rowan 11986156	Document creation and initial population
<b>2018-11-14T10:05:06Z</b>	Rowan 11986156	Add LED Power Test
<b>2018-11-24T08:12:06Z</b>	Sanket	3.2.4 - Gesture Control Verification
<b>2018-11-16T09:13:13Z</b>	Rowan 11986156	LED Text Display Test
<b>2018-11-20T12:29:13Z</b>	Jimmy 40243140	Population of I2C Testing Section
<b>2018-11-25T16:41:29Z</b>	Jimmy 40243140	Small edits and formatting
<b>2018-10-25T18:42:59Z</b>	Rowan 11986156	<ul style="list-style-type: none"><li>• IR Emitter/Sensor Selection Verification test</li><li>• Add IR Sensor Object Detection test</li></ul>
<b>2018-11-25T20:35:37Z</b>	Parth	Added Linear regulator functionality test
<b>2018-11-25T20:35:37Z</b>	Candice 16550155	Section 4.2 - Power converter simulation Minor format revisions
<b>2018-11-25T20:35:37Z</b>	Rowan 11986156	Milestone 2 prep work
<b>2019-02-10T19:19:04Z</b>	Jimmy 40243140	Milestone 3 review, Update with Topology Test

# Table of Contents

1 - I <sup>2</sup> C Communication Tests .....	1
1.1 - Common Setup .....	1
1.2 - I <sup>2</sup> C Scan Test .....	2
1.3 - I <sup>2</sup> C Dynamic Addressing .....	4
1.4 - I <sup>2</sup> C Dynamic Display.....	6
1.5 - I2C Communication Testing Results .....	8
2 - LED Tests .....	9
2.1 - Common Setup .....	9
2.2 - PCB LED Functionality Verification .....	9
2.3 - LED Power Consumption .....	11
2.4 - Text Display .....	13
3 - IR Sensor and IR Mask Tests .....	14
3.1 - Common Setup .....	14
3.2 - PCB IR Sensor Functionality Verification .....	15
3.3 - IR Sensor Object Detection.....	18
3.4 - IR Emitter and Sensor Selection Justification .....	19
3.5 - Gesture Control Verification .....	21
4 - Power Components Verifications.....	22
4.1 - Linear regulator functionality test .....	22
4.2 - Power Converter Design Simulation .....	23
5 - Topology Tests .....	27
5.1 - Position Mapping.....	27
Appendix I: Testing Source Code .....	29
Appendix II: Pinout Diagrams .....	57
Citations .....	59

## Table of Figures

Figure 1: I2C Simple Data Transfer Test Schematic .....	2
Figure 2: I2C Dynamic Addressing Test Schematic .....	4
Figure 3: I2C Dynamic Display Test Setup .....	6
Figure 4: LED Test Connection Schematic.....	9
Figure 5: Steady State Sensor Output - Ambient Sunlight Condition .....	16
Figure 6: Steady State Sensor Output - Direct Sunlight .....	16
Figure 7: IR Emitter and Sensor Pair Irradiance diagram .....	19
Figure 8: Expected Reflected Irradiance vs Distance .....	20
Figure 9: AC/DC and DC/DC Buck Converter Circuit Schematic .....	24
Figure 10: Power Converter Simulation Waveforms and Average Values.....	25
Figure 11: Sample Digikey BOM for Power Converter Components for 10,000 Tiles .....	26
Figure 12: Topology Test Schematic .....	27

## Table of Tables

Table 1: Current Draw of APA102 2020 LEDs at Different Brightness Levels.....	12
Table 2: Specifications of the Designed Buck Converter for a Single Tile .....	23

# 1 - I<sup>2</sup>C Communication Tests

These tests demonstrate the data communication between multiple Tiles and verify features available within the protocol.

## 1.1 - Common Setup

Two STM32 devices are connected to the I<sup>2</sup>C bus operating at 100kHz. 4.7 kΩ resistors are used to pull the logic level high for SCL and SDA lines of the I<sup>2</sup>C bus. The standard I<sup>2</sup>C pins for the STM32F103C8 (SCL as PB6 and SDA as PB7) are connected to the microcontroller I<sup>2</sup>C bus.

The 3.3V rail of the bootloader connects and powers the microcontrollers to simulate the devices being powered from a single power source. Functions from the I<sup>2</sup>C “Wire” library initializes and configures the bus for use with the microcontroller.

## 1.2 - I<sup>2</sup>C Scan Test

### Purpose

- To verify that the master device acknowledges the presence of one a slave device on the bus.
- To transfer data from the master device to slave over the I<sup>2</sup>C bus.

### Setup

[Fig 1] shows the test setup as described in section 1.1. The slave device will have a hard coded of address 0x02.

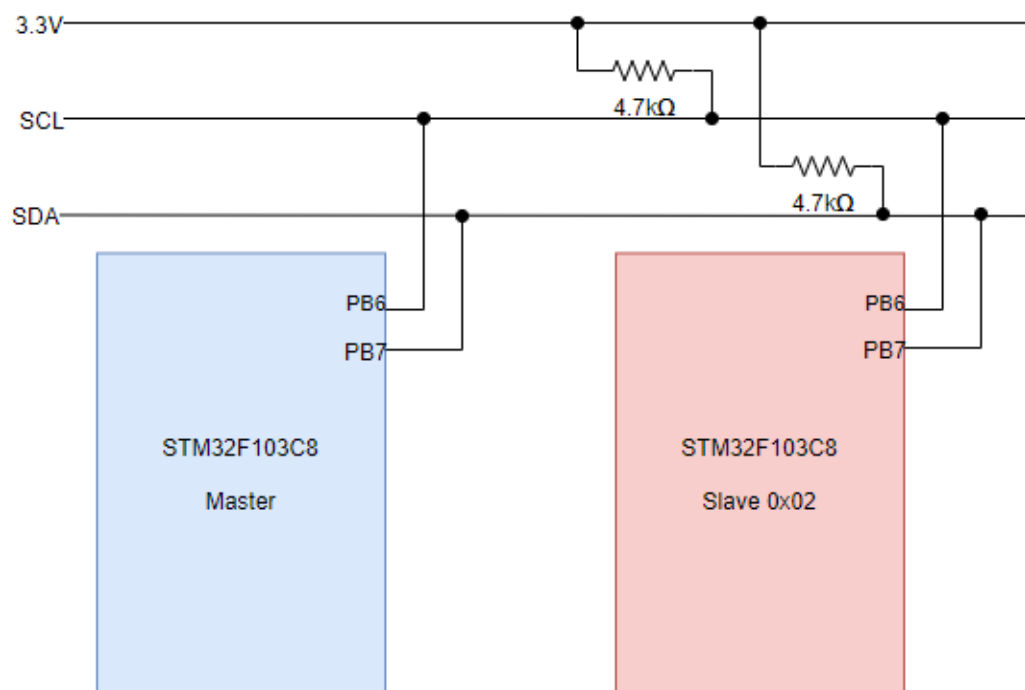


Figure 1: I2C Simple Data Transfer Test Schematic

### Procedure

Master Code: [i2c\_LED\_Master.ino, Appendix I]

Slave Code: [i2c\_LED\_Slave.ino, Appendix I]

1. Initialize I<sup>2</sup>C on master device.
2. Initialize I<sup>2</sup>C on slave device with address 0x02
3. Master transfers 1 byte of data if the slave is available

- a. The transferred data is a counter modulo 2 (alternating between 0 and 1)
4. Slave turns the built-in LED on and off to reflect value based on the last value received.

## Outcome

The master can successfully detect and communicate with the slave device. The slave successfully receives the byte and can blink the LED about 2 times a second. This confirms data transfer is successful on the I2C bus with one Tile. This can also be extended to three Tiles if we initialize another slave device with a different address.

## 1.3 - I<sup>2</sup>C Dynamic Addressing

### Purpose

- To test dynamic allocation of the address of the slave devices.

### Setup

[Fig 2] shows the test setup as described in section 1.1 with some additions. A push button connected to the slave device at pin PA5 resets the device address when pressed.

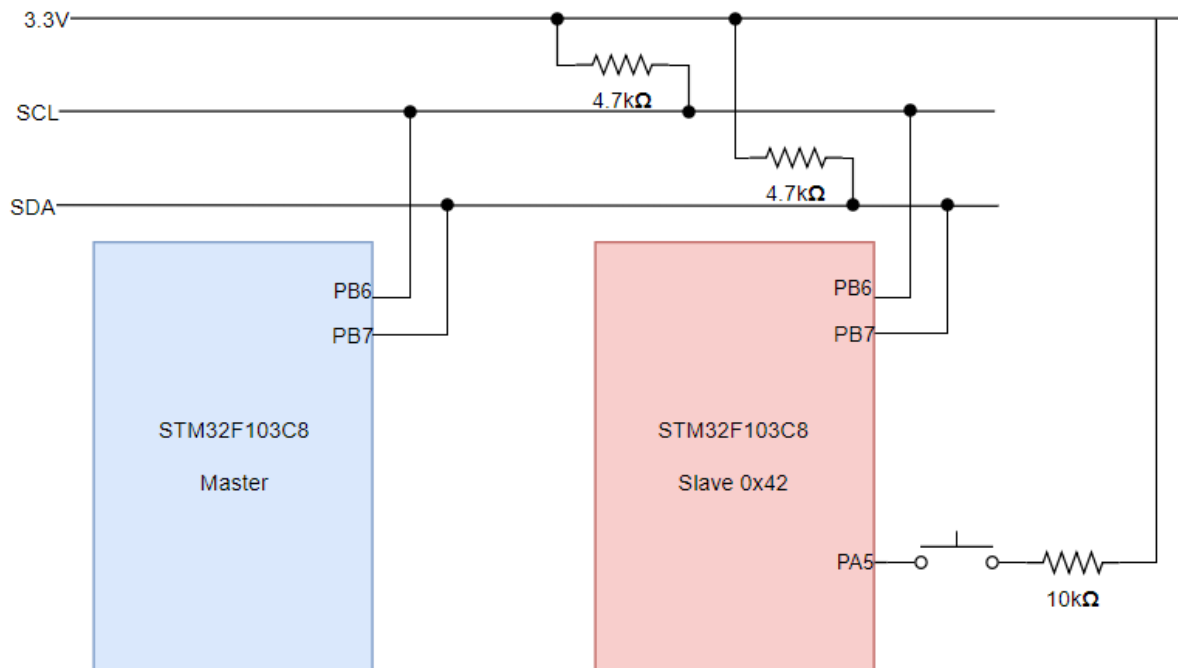


Figure 2: I2C Dynamic Addressing Test Schematic

### Procedure

Master Code: [I2C\_DynamicAddress\_Master.ino, Appendix I]

Slave Code: [I2C\_DynamicAddress\_Slave.ino, Appendix I]

1. Initialize I<sup>2</sup>C on master device.
2. Initialize I<sup>2</sup>C on slave device using default address 0x42.
3. Master scans the bus for all addresses
  - a. The master sends the slave device one-byte data containing new address if a device is detected at the default address. This test uses 0x6A as the new address.



4. Slave device holds this new address in a variable until a push button is pressed. Upon pressing the button, slave restarts the I<sup>2</sup>C port with the new address.
5. Master device now sees only one device at address 0x6A.

## Outcome

At the beginning of the test, only one device at address 0x42 (default address) on the I<sup>2</sup>C bus. After pushing the button there is still only one device found on the I<sup>2</sup>C bus by the master. However, the address is 0x6A which is the address that the master sent to the slave with the default address of 0x42 earlier. The results of this test prove we can re-initialize slave devices using dynamic address allocation.

## 1.4 - I<sup>2</sup>C Dynamic Display

### Purpose

- To verify the ability to add and remove devices from the I<sup>2</sup>C bus dynamically

### Setup

[Fig 3] shows the test setup as described in section 1.1 with some additions. This test also includes 4 LEDs connected to each Tile. The LEDs are connected to pins PA1-4.

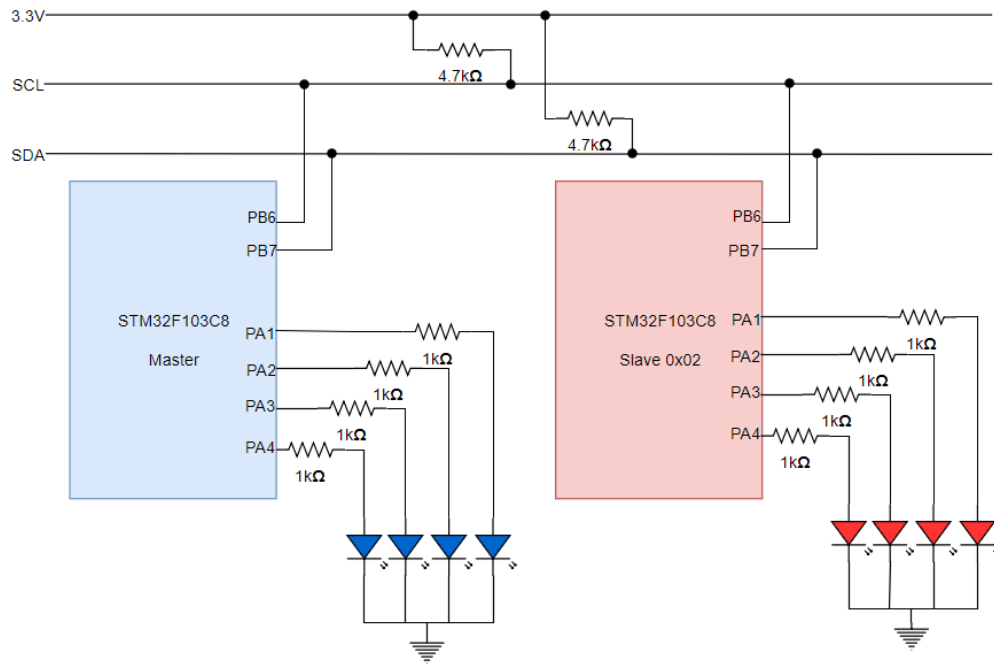


Figure 3: I2C Dynamic Display Test Setup

### Procedure

Master Code: [I2C\_DynamicDisplay\_Master.ino, Appendix I]

Slave Code: [I2C\_DynamicDisplay\_Slave.ino, Appendix I]

1. Initialize I<sup>2</sup>C on master device with LEDs 1 to 4.
2. Initialize I<sup>2</sup>C on slave device using default address 0x02 and LEDs 5 to 8.
3. Master determines which LEDs to turn on depending on the number of devices on the bus.

- a. The master scrolls the LED across all LEDs currently connected to the display in an endless loop. Both the red and blue LEDs turn on if both devices are connected, otherwise only the blue LEDs turn on.
4. Slave listens on the bus to determine the output value of the red LEDs.
5. Occasionally, the reset button on the slave device is pressed to simulate removing it from the bus. Letting go of the reset button will simulate adding the device to the bus.

## Outcome

Video of the test can be seen here:

[https://drive.google.com/open?id=1nmlbj\\_mVQLO1qjlOnqgz28PdHdXJ0SXw](https://drive.google.com/open?id=1nmlbj_mVQLO1qjlOnqgz28PdHdXJ0SXw)

Operating at about 10 fps we are able observe the LEDs dynamically changing the length of the display when the slave is “added and removed”. This test verifies that we can add and remove devices from the bus dynamically.

## 1.5 - I2C Communication Testing Results

Slave device addresses can change after the device has already been turned on. This allows us to keep the slave source code the same and have the master device reassign an address instead of using a predetermined device address.

## 2 - LED Tests

### 2.1 - Common Setup

[Fig 4] shows the connection setup for the LED Tests. For detailed pin out diagrams of the STM32 dev board and test PCB please see [Fig 1, Appendix II] and [Fig 2, Appendix II] respectively.

1. Connect the PB10 and PB11 pins of the STM32 to the PCB's DATA INPUT and CLOCK INPUT pins respectively
2. Supply the PCB with 3.3V from a separate power supply
3. Ensure grounds of the PCB and STM32 are connected to minimize signal noise
4. Boot load the relevant testing code to the STM32

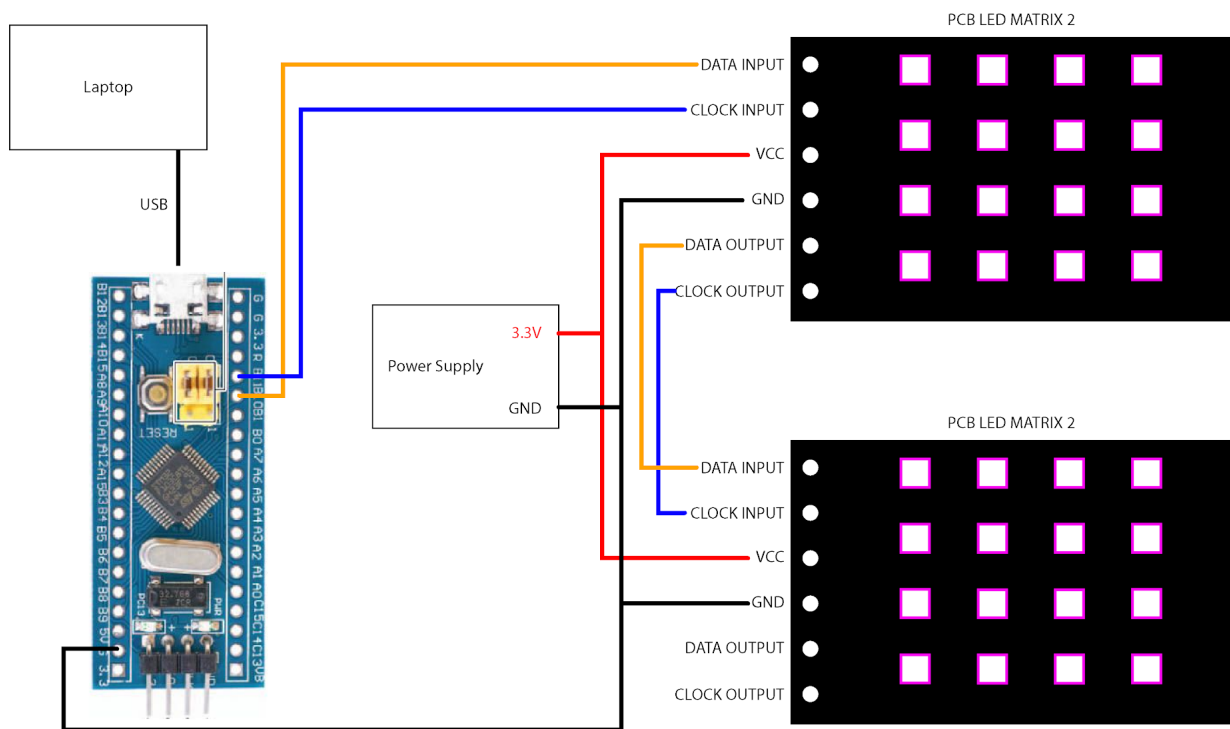


Figure 4: LED Test Connection Schematic

### 2.2 - PCB LED Functionality Verification

#### Purpose

- To check each LED on the test PCB for defects.

## Setup

White light is an even combination of red, green, and blue light, and ideal for visual verification that all colors are functioning for a given LED. The white light is set with a maximum of  $\frac{1}{4}$  brightness so as not to blind the tester.

## Procedure

STM32 Code: [MatrixTestStm32.ino, Appendix I]

1. Cycle every LED with white light using testEachPixel()

## Outcome

Three of the four PCBs have a fully functioning matrix of LEDs. The faulty PCB functions only for the first 7 serially connected LEDs. This is likely due to an error during the manual placement of parts on the PCB.

## 2.3 - LED Power Consumption

### Purpose

- To determine the worst-case power consumption of a 8x8 LED matrix.

### Setup

The PCB is powered with 3.3 Volts and the current draw is recorded for each color and maximum brightness cycle.

### Procedure

STM32 Code: [MatrixTestStm32.ino, Appendix I]

1. The testAllPixel() turns on all 16 LEDs simultaneously and cycles them through white, blue, green, and red light at full,  $\frac{1}{2}$ ,  $\frac{1}{4}$ ,  $\frac{1}{8}$ , and  $\frac{1}{16}$  maximum brightness settings.

A video of the test can be seen here:

<https://drive.google.com/file/d/1RRLvuoGjAmrLjJZLJ2IU8eCijgDMBpZ/view?usp=sharing>.

## Outcome

[Table 1] lists the current draw of the PCB at varying brightness levels and colours.

Table 1: Current Draw of APA102 2020 LEDs at Different Brightness Levels

	Color				Worst Case Power (4x4 Matrix)	Worst Case Power (8x8 Matrix)
Max Brightness	White	Blue	Green	Red	3.3V*(White light Current)	4*(Worst Case Power (4x4 Matrix))
Full	135mA	50mA	54mA	90mA	.45W	1.8W
1/2	84 mA	34mA	36mA	55mA	.28W	1.12W
1/4	53mA	25mA	26mA	36mA	.17W	.68W
1/8	34mA	20mA	21mA	26mA	.11W	.44W
1/16	25mA	18mA	19mA	20mA	.083W	.332W

The LEDs are too bright to look at directly for max brightness settings of Full,  $\frac{1}{2}$ , and  $\frac{1}{4}$ . This suggests the LEDs can operate with a maximum brightness of  $\frac{1}{8}$ , which has a worst-case power consumption of .68W and current draw of 206mA for an 8x8 LED matrix.

## Design Ramifications

During testing it was discovered that there are two different LEDs on three of the PCBs. One set of LEDs is significantly dimmer than the other at equivalent power consumptions. After confirming the LEDs had the same part and batch number, it was determined there is an issue with the current LED supplier. Further LEDs will be procured from the Adafruit manufacturer, as they provide the same LED part number at a similar price with top review ratings.

## Further Testing

The LED light will also be diffused through a square piece of glass to provide a larger pixel display and more even lighting effect. Because of this and the manufacturing anomaly mentioned earlier, higher brightness, and therefore more power, may be required than currently assumed.



## 2.4 - Text Display

### Purpose

- To demonstrate that text can be displayed and scrolled across the LED matrices.
- To verify the matrices can be linked together for larger displays.
- To profile the processing time required to display text on a LED matrix.

### Setup

The 4x8 pixel display can only show the top half of characters so the string “” ^ ~” was chosen.

Configuration of a single display was achieved by setting the following variables:

- `matrixWidth = 8;`
- `matrixHeight = 4;`

Simulation of a full 8x8 Tile matrix was achieved by setting the following variables:

- `matrixWidth = 8;`
- `matrixHeight = 8;`

The STM32 MCU processing time is recorded by setting a pin HIGH at the beginning of the `scrollText()` function, and LOW upon function completion. An oscilloscope is used to measure the interval between the pins high and low output.

### Procedure

STM32 Code: [MatrixTestStm32.ino, Appendix I]

1. The `scrollText()` function scrolls a string from left to right on three LED matrices connected to make a 4x8 pixel display.

A video of the test can be seen here:

<https://drive.google.com/file/d/1BRqLP1Gf0W6TVumukrwBBzP3Aoo1r-3-/view?usp=sharing>

### Outcome

Text can be successfully scrolled across the matrixes. There is no reason to assume this cannot be scaled to a full 8x8 Tile matrix.

For an 8x4 LED Matrix, the `scrollText()` function required 1.25ms of MCU time per frame. For an 8x8 LED Matrix, the `scrollText()` function required 10ms of MCU time per frame. This gives a maximum frame rate of 100 fps assuming 100% of MCU time is spent updating the LED matrix. As our target frame rate is 24fps [1], this would require approximately 24% of MCU time be devoted to matrix updating.

## 3 - IR Sensor and IR Mask Tests

### 3.1 - Common Setup

Initial setup should be the same as the LED tests: Common setup, refer to [Fig 5] if needed. The STM32 dev board used for testing has only 10 analog pins capable of being read by the ADC. Thus 10 of the 16 sensors are read using the analog pins PA0-PA8, PB0, and PB1. Sensors S7-S8 are mapped to pins PB1 and PB0 respectively while sensors S9-S16 are mapped to pins PA8-PA0 respectively, see [Fig 2, Appendix II] and [Fig 1, Appendix II] for a detailed pin map of the STM32 dev board and test PCB. Each sensor value was processed using the ADC. All sensors were polled 10 times every second using an interrupt subroutine. The values are then printed to the serial monitor.

## 3.2 - PCB IR Sensor Functionality Verification

### Purpose

- To determine IR sensor sensitivity to Background IR levels.
- To determine the effectiveness of the IR Mask.

### Setup

An oscilloscope was connected to the output of several sensors not measured by the ADC. The IR mask used is 1cm thick, with a 1mm diameter hole centered above the IR sensor. Testing was conducted around noon on a sunny day.

### Procedure

STM32 Code: [IRSensorsSTM32.ino, Appendix II]

1. Connect the oscilloscope probe to the output of the IR Phototransistor.
2. Begin the code
3. Measure the phototransistor voltage in normal room-light conditions without sunlight (curtains drawn)
4. Measure the voltage of the phototransistor with a heavily sun lit room (curtains opened)
5. Repeat steps 1 - 4 with the IR Mask.

### Outcome

The sensors output less than 1mV in room light conditions with the currents drawn, which registers a raw analog to digital converter (ADC) value of 0. The sensors output roughly 50 mV in room light conditions with the currents up [Fig 6], which gives measurable A values around 60. This leads to a condition with insufficient IR to detect movement in normal room lighting conditions, and too much IR to detect movement in direct sunlight. With the IR mask attached to the sensor matrix, the sensors are unable to output more than 150 mV when facing direct sunlight [Fig 6].

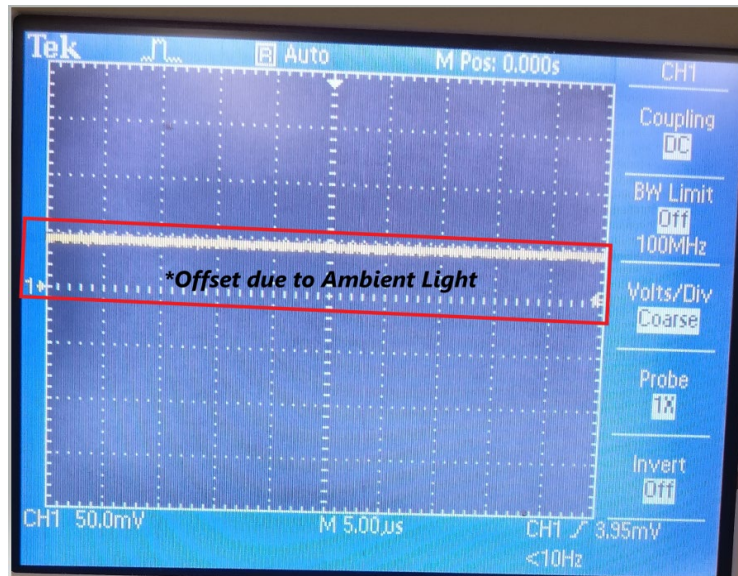


Figure: Steady State Sensor Output - Ambient Sunlight Condition

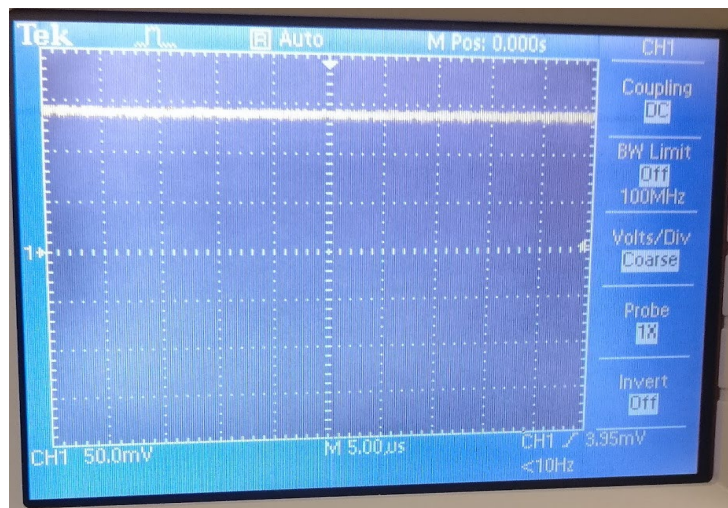


Figure 5: Steady State Sensor Output - Direct Sunlight

## Design Ramifications

IR emitters will have to be added in order to generate IR that can be reflected off objects in front of the sensor matrix.

The IR mask will have to allow more light to reach the sensors or be removed entirely. Furthermore, the primary purpose of the IR mask will be to shield the sensors from direct sunlight.

Even when not in direct sunlight, the IR sensors still read non-zero values from ambient sunlight on sunny days.

## Suggested Solution

IR sensors with daylight filtering lenses, such as the VEMT2020x01, should be used.

## 3.3 - IR Sensor Object Detection

### Purpose

- Demonstrate that an object can be detected by measuring reflected IR
- Verify real-time LED matrix updating

### Setup

STM32 Code: [IRSensorsSTM32.ino, Appendix II]

1. Set up an array of IR LEDs next to the IR sensor matrix.
2. Run the sensor polling code as described in the common setup.
3. Map each sensor value from 0 to 4096 to 0 to 255 and store it in an array.
4. Threshold the value to account for IR noise.
5. Pass the value as a brightness to the sensor's nearest LED.

### Procedure

A tester moves an object near the sensor matrix to be detected.

A video of the test can be seen here:

[https://drive.google.com/file/d/1QyKGC2f2Skh5RIxxfqVEEDQ5djhL\\_1oR/view?usp=sharing](https://drive.google.com/file/d/1QyKGC2f2Skh5RIxxfqVEEDQ5djhL_1oR/view?usp=sharing)

### Outcome

The sensors can see reflected IR. Furthermore, the LED matrix can be updated real time without issue. However, as the sensors are located off the PCB and there is no IR mask obscuring the sensors from each other, the sensors see a “blurry” object.

### Design Ramifications

The IR emitters should be paired with and located as close as possible to the IR sensors to reduce IR noise and ensure strong but local IR levels. Furthermore, if the IR sensor/emitter pairs can be turned on and off during the sensor polling interrupt routine, the measured IR value will not be influenced by other IR sensor/emitters pairs. See the “Updated IR sensor Matrix Design” section in the design document for further details.

### 3.4 - IR Emitter and Sensor Selection Justification

#### Purpose

- Determine if the VEMT2020x01 NPN Phototransistor will meet project requirements
- Determine if the SFH 4056 Infrared Emitter will meet project requirements

#### Setup

The phototransistor detects objects by measuring the amount of IR reflected from them. To determine the range of detection we first estimate the irradiance at the sensor as a function of distance from the emitter. The following equation solves for irradiance at the sensor:

$$L_{sensor} = \frac{(RF_{object}) L_{emitter}}{D^2}$$

For a more intuitive understanding of what these variables relate to, see [Fig 7].

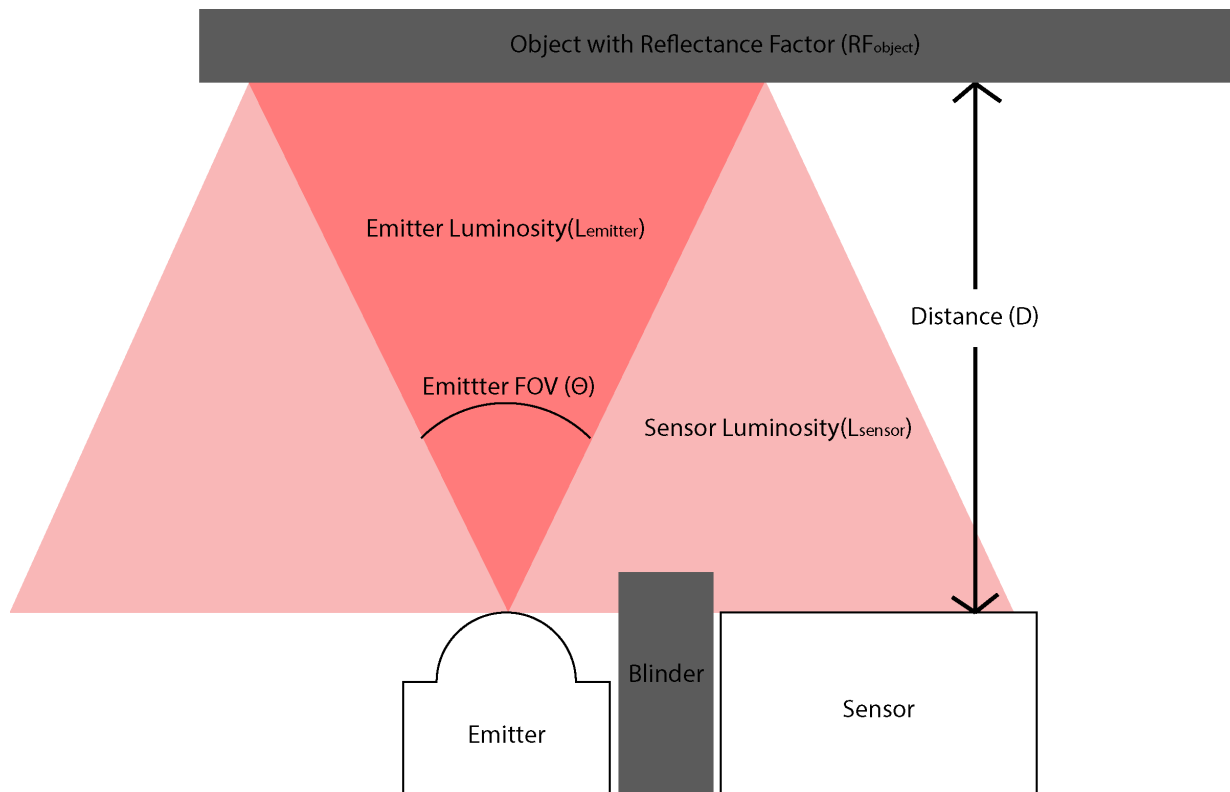


Figure 6: IR Emitter and Sensor Pair Irradiance diagram

## Procedure

The expected irradiance at the sensor as a function of distance is graphed using the following assumptions:

1. The IR emitter is the Osram SFH 4056 with a minimum irradiance @70mA is  $2.5\text{mW/cm}^2$  [1].
2. The reflectance factor of human skin ( $RF_{\text{object}}$ ) is approximately 0.55 [2].

The graph is generated with a minimum irradiance of  $2.5\text{mW/cm}^2$ ,  $4.25\text{mW/cm}^2$ , and  $6\text{mW/cm}^2$ . These values can be achieved by supplying the IR emitter with a forward current of 70mA and are dependent on the binning of the device [1].

## Outcome

[Fig 8] compares reflected IR irradiance at the sensor with object distance from the sensor.

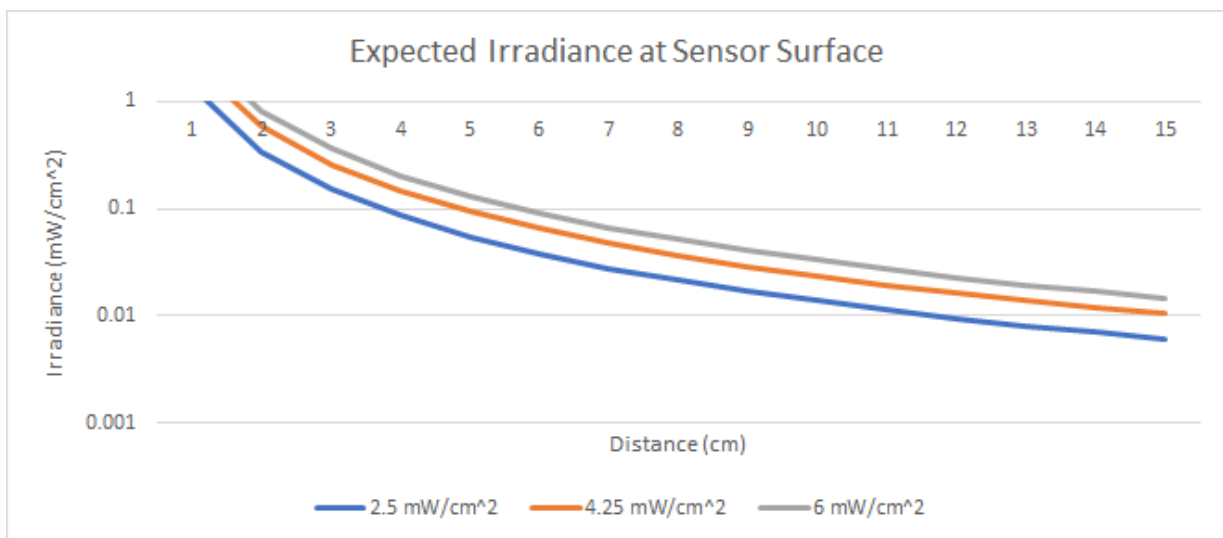


Figure 7: Expected Reflected Irradiance vs Distance

The above graph suggests there is enough IR reflected from a hand up to 11 cm away to read with the VEMT2020x01 IR phototransistor, as required in by requirement 6.5.2. This phototransistor can detect irradiances as low as  $.01\text{mW/cm}^2$  [3].



## 3.5 - Gesture Control Verification

### Purpose

- To prove the gesture control abilities of the APDS 9960 module.

### Setup

The 9960 module is a pre-packaged module, with a built in IC and 4 photodiodes, used to determine the direction of the user's hands movement.

STM32 Code: [*GestureControl\_LED.ino* code, in Appendix I]

1. Connect the 9960 module via the I2C pins on board to the STM32.
2. Send sensor readings from the 9960 to the STM32.
3. Output sensor data from STM32 to turn the LEDs on.

### Procedure

After the setup the test was conducted by a user maintaining a distance of 4 inches of the board (optimal reading range recommended by the 9960-module datasheet) and waving their hand arbitrarily in one of the four directions (Up, Down, Left, Right). The Serial Monitor on the Arduino IDE would output the direction the user waved their hand and a specific colour on the LEDs depending on direction.

### Outcome

When the user waves their hand in one of the distinct directions, UP, DOWN, LEFT, RIGHT, it corresponds to a specific LED colour change (UP = Green; DOWN = RED; LEFT = Blue; RIGHT = WHITE). This is a starting point and a proof of concept to show the output of 9960 module is capable of the controlling the test PCB.

## 4 - Power Components Verifications

### Purpose

To prove physical functionality of components selected, as well as feasibility of the designed power system from wall to each component.

### 4.1 - Linear regulator functionality test

#### Purpose

- To test an available linear voltage regulator that outputs the required 3.3V using variable higher voltage input to see how the input current varies based on input voltage.

#### Setup

For testing purposes, we used MCP1700- 3302E/TO linear regulator which gives our desired 3.3V along with a maximum current rating 250mA [9], which was enough for our test PCB (1/16th of a final Tile size). We connected the output to a 51Ω resistor and an adjustable DC power supply on the input. A voltmeter was connected across the resistor to observe the voltage.

#### Procedure

Current requirement for resistor used:

$$I = V / R = 3.3/51 = 65\text{mA}$$

We supplied the 1V- 5V voltage to the resistor via the regulator and observed steady voltage output across the resistor which varied with the  $V_{in}$  up to 3.3V and stayed at 3.3V at all  $V_{in}$  above that. The current drawn was recorded at multiple  $V_{in}$ 's.

#### Outcome

The current drawn remained stable at 65mA at anything above  $V_{in} = 3.3\text{V}$ , therefore telling us that all the increase in power was being lost. The linear regulator is not able to regulate current when the supplied voltage is in excess hence all the excess power input (from increased voltage) is lost as heat. This way the regulator draws all its output current directly from source.

The heat level significantly increased at  $V_{in} > 6\text{V}$  and was too hot to even touch above 9V input, which can be calculated as:  $[(V_{in} - V_{out}) \times I_{out}]$  Watts

Additionally, there was a dropout voltage ( $V_{out} - V_{in}$  for all  $V_{in}$  up to regulator limit) of 0.2V.

#### Design Ramifications

For our power supply of 18V, the heat dissipation would be very high and hence we needed to lower the difference between the  $V_{in}$  and  $V_{out}$  at the linear regulator. For that reason, we will be utilizing a DC-DC buck convertor set around 4V to compensate for dropout voltage while keeping heat dissipation minimum.

## 4.2 - Power Converter Design Simulation

### Purpose

To demonstrate the feasibility of designing a 40-60W power converter in the interest of reducing cost. This is a consideration for the client for future iterations of the product.

### Setup

The power converter design considers the wall outlet as the source, and thus, should have a rectifier, some form of isolation, and a step-down (buck) converter. The buck converters have specifications as outlined in [Table 2] below. Note that stepping down the voltage should be done in two stages because the total difference between the input and output voltage is very large.

Table 2: Specifications of the Designed Buck Converter for a Single Tile

Design Specifications	Input Voltage	Output Voltage	Input Power	Inductor Current Ripple	Voltage Ripple
Stage 1	120V <sub>rms</sub>	30V	7.8W	---	±.01%
Stage 2	30V	h3.3V	7.8W	±8%	±.5%

Simulations were performed in PSIM using the circuit shown in [Fig 9]. Component values were chosen according to the calculations in the procedure section.



## Outcome

The simulation results in [Fig 11] confirm that all design specifications are met for one Tile.

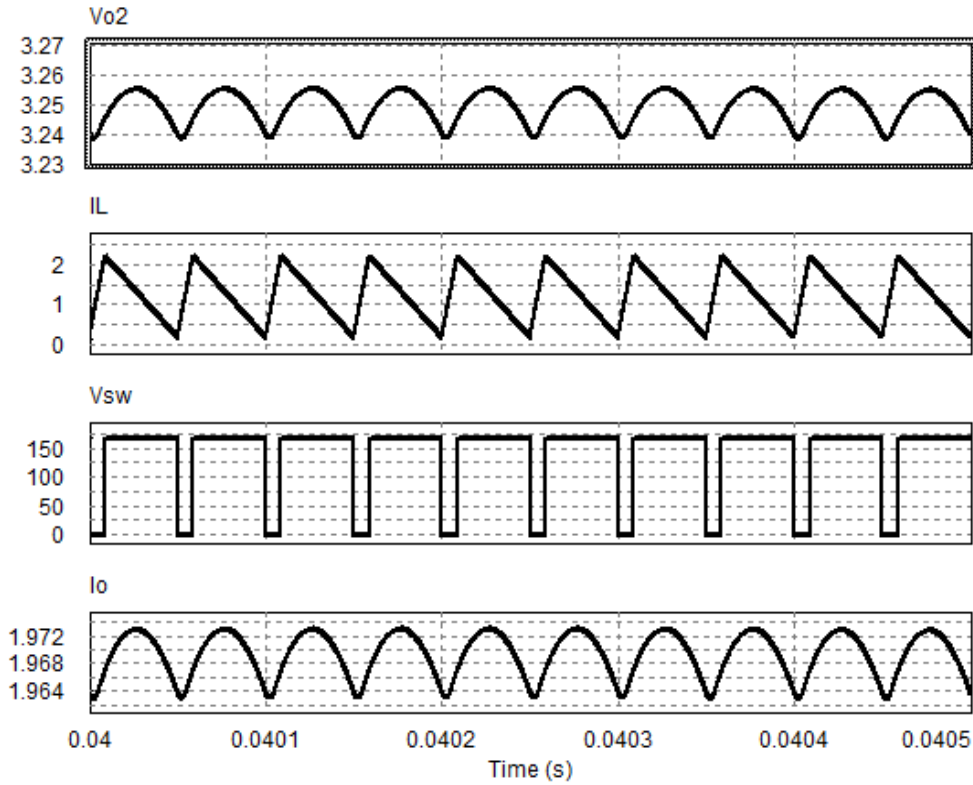


Figure 9: Power Converter Simulation Waveforms and Average Values

Vo2	3.2494760e+000
Io	1.9693794e+000

$$\Delta V_o = 16.5364\text{mV} < .16V_o$$

Implementing the buck converters in two stages allows us to transmit power between Tiles at a lower current, and thus, use fewer pogo pins because stage 1 will be part of the AC/DC converter on the master Tile and stage 2 will be on every Tile. To power a full display, each Tile's buck converter (stage 2) will be interleaved, which has the added benefit of being more efficient than a single buck converter [4]. By interleaving the buck converters, we are essentially performing a current step-up at the load, as the loads add in parallel to draw more current from stage 1 up until 10 Tiles are connected.

Note that with this configuration, boundary discontinuous conduction mode (DCM) for the first stage should be considered because a resistive load must be connected in parallel with the output capacitor. The resistive load at the output of the first stage dissipates minimal power when it has its maximum value [5]. This is computed below.

$$R_{LI} = \frac{2f_s L}{1 - D} = 30\Omega$$

The simulation shows this value to be correct because in the inductor current waveform of the first stage, the trough reaches nearly zero, but does not reach DCM ( $i_L(t) = 0A$ ).

While the cascaded buck converter configuration works, it is not the most efficient power converter design. Integrating the transformer into the converter is possible in more complex topologies such as the fly back or half-bridge converter and can reduce the number of components used in the design, but due to the scope of this capstone project, it will suffice to say that designing a power converter for use with the LED and IR matrices is possible.

In a sample parts order [Fig 12] the total cost of implementing self-designed power converters on 10,000 boards is CAD\$14,857.37, or CAD\$1.49 per Tile, and CAD\$14.86 for every 10 Tiles. This price is comparable to a very cheap computer charger but has the benefits of being more efficient (i.e. the input to the voltage regulator will be designed to more closely match the voltage output of the regulator, reducing power losses in the voltage regulator, as described in section 4.1) and can be integrated into the packaging of each Tile. One major drawback is that this design would need to be approved by regulatory bodies before being made commercially available. This will consume a large amount of time, so we recommend that this design be used in a future iteration of the product.

Index	Quantity	Part Number	Description	Unit Price	Extended Price CAD
1	11000	495-3851-ND	FERRITE CORE TOROID 5.11UH T38	0.24174	2,659.18
2	2000	P10396TB-ND	CAP ALUM 47UF 20% 50V RADIAL	0.0794	158.79
3	10000	1189-2244-ND	CAP ALUM 68UF 20% 25V RADIAL	0.07756	775.64
4	1000	IPN60R3K4CEATMA1CT-ND	MOSFET NCH 600V 2.6A SOT223	0.26156	261.56
5	1020	VPP10-1000-B-ND	XFRMR LAMINATED 10VA THRU HOLE	8.51217	8,682.41
6	1000	RMCF0402JT30R0CT-ND	RES 30 OHM 5% 1/16W 0402	0.00338	3.38
7	1000	1655-1807-1-ND	BRIDGE RECT 1PHASE 600V 2A ABF	0.14577	145.77
8	9000	SSM3K339RLFTR-ND	MOSFET N-CH 40V 2A SOT-23F	0.12461	1,121.46
9	1000	SSM3K339RLFCT-ND	MOSFET N-CH 40V 2A SOT-23F	0.1476	147.60
10	10000	MBR2H200SFT3GOSTR-ND	DIODE SCHOTTKY 200V 2A SOD123FL	0.09016	901.58
				<b>Total Price</b>	<b>14,857.37</b>

Figure 10: Sample Digikey BOM for Power Converter Components for 10,000 Tiles

## 5 - Topology Tests

### 5.1 - Position Mapping

#### Purpose

- To verify that the master can map out the Tiles using the four directional pins
- To verify that the master can reorder the Tiles based their relative locations.
- To verify that the master can send different data to different Tiles.

#### Setup

The setup [Fig 12] builds the common setup described in section 1.1. The 4 x 4 Test PCB is connected to each microcontroller on pins PB11 and PB10 to transmit serial data. Not shown are four arbitrary pins used to determine topology. The pins used in the source code to represent the four directions are PA3, PA4, PA5 and PA6. [TopologyTest\_Master.ino, Appendix I]

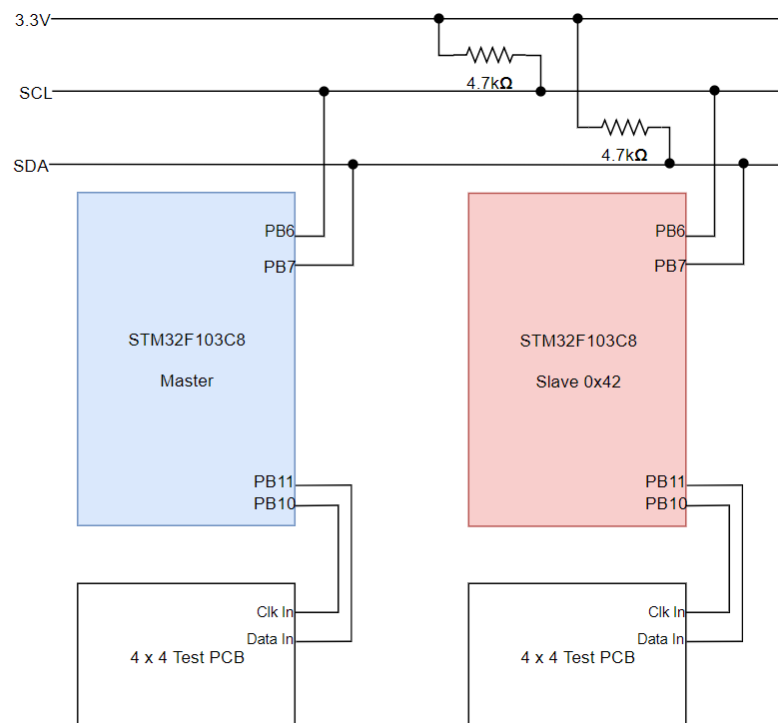


Figure 11: Topology Test Schematic

## Procedure

Master Code: [TopologyTest\_Master.ino, Appendix I]

Slave Code: [TopologyTest\_Slave.ino, Appendix I]

1. Power up the master device and connect the I<sup>2</sup>C pins of a second device.
2. Connect the directional pin corresponding to the down direction to 3.3V on the master. Then power up the slave using 3.3V
3. The 4 x 4 test PCB connected to the master should display a 2x2 green box while the test PCB connected to the slave displays blue.
4. Verify on the serial output that the displayed array shows the master Tile at the center and the slave Tile (#4) is directly below.
5. Turn off the slave device and change the powered directional pin corresponding to the up direction instead of down. Power the slave device again.
6. The 4 x 4 test PCB connected to the master should now display a 2x2 blue box while the test PCB connected to the slave displays green.
7. Verify on the serial output that the displayed arrow shows the master Tile at the center and the slave Tile (#4) is directly above.
8. Repeat with another slave using the Left and Right directional pins.
9. Verify the serial output displays the correct order. The uppermost Tile takes priority with a preference for the Tile closest to the left.
10. Check the colours displayed by the test PCB connected to each Tile.

## Outcome

The master can correctly map out the devices in the internal array and is able to reorder them when a device is added or removed. The master is then able to send data according to the order to each of the slave devices. The test PCB connected to each Tile also reflects this change. The order of the Tiles determines the colour displayed with the first Tile showing green, the second showing blue, and the third showing red.



# Appendix I: Testing Source Code

The following source code was used in the test procedures outlined in this document.

## A1.1 - GestureControl\_LED.ino

```
#include <Adafruit_GFX.h>
#include <Adafruit_DotStarMatrix.h>
#include <Adafruit_DotStar.h>
#include "Adafruit_APDS9960.h"
Adafruit_APDS9960 apds;

// Pin setup
const uint8_t MATRIX_DATA_PIN = 4;
const uint8_t MATRIX_CLK_PIN = 7;

// Size of each Tile matrix
const uint8_t matrixWidth = 4;
const uint8_t matrixHeight = 4;

// Number of Tile matrices
const uint8_t TilesX = 1;
const uint8_t TilesY = 1;

Adafruit_DotStarMatrix matrix = Adafruit_DotStarMatrix(
  matrixWidth,
  matrixHeight,
  TilesX,
  TilesY,
  MATRIX_DATA_PIN,
  MATRIX_CLK_PIN,
  DS_MATRIX_TOP + DS_MATRIX_LEFT +
  DS_MATRIX_COLUMNS + DS_MATRIX_ZIGZAG + DS_TILE_PROGRESSIVE,
  DOTSTAR_RGB
);

const uint16_t colors[] = {
  matrix.Color(255, 0, 0), matrix.Color(0, 255, 0), matrix.Color(0, 0, 255), matrix.Color(255, 255, 255), matrix.Color(0,0,0)
};

void setup() {
  #if defined(__AVR_ATtiny85__) && (F_CPU == 16000000L)
    clock_prescale_set(clock_div_1); // Enable 16 MHz on Trinket
  #endif

  matrix.begin(); // Initialize pins for output
  matrix.setBrightness(45); // Set max brightness (out of 255)

  matrix.setTextWrap(false);
  matrix.setTextColor(colors[0]);
  matrix.show(); // Turn all LEDs off ASAP
  Serial.begin(115200);

  if(!apds.begin()){
    Serial.println("failed to initialize device! Please check your wiring.");
  }
}
```

```

else Serial.println("Device initialized!");

//gesture mode will be entered once proximity mode senses something close
apds.enableProximity(true);
apds.enableGesture(    true);

}

void testAllPixels(uint8_t index) {
    matrix.setCursor(0, 0);
    matrix.fillRect(0, 0, matrixWidth, 4*matrixHeight, colors[index]);
    matrix.show();
}

void loop() {

    //read a gesture from the device
    uint8_t gesture = apds.readGesture();

    switch (gesture){

        case(AP_DS9960_DOWN):
            testAllPixels(2);
            Serial.println("DOWN");
            delay(1000);
            break;

        case(APDS9960_UP):
            testAllPixels(1);
            Serial.println("UP");
            delay(1000);
            break;

        case(APDS9960_LEFT):
            testAllPixels(0);
            Serial.println("LEFT");
            delay(1000);
            break;

        case(APDS9960_RIGHT):
            testAllPixels(3);
            Serial.println("RIGHT");
            delay(1000);
            break;

        default:
            testAllPixels(4);
            break;

    }

}

```

## A1.2 - I2C\_DynamicAddress\_Master.ino

```

/*****
*           Title: I2C_DynamicAddress_Slave

```

```

*      Description: Modification of i2c_scanner example for STM32
*      Author: Jim my Wong
*      Date: Nov 18, 2018
*      Code version: 0.0.1
*****
*****/

#include <Wire.h>
#define I2C_DEFAULT 0x42

void setup() {

    Serial.begin(9600);
    Wire.begin();
    Serial.println(" \ nI2C Scanner");

}

void loop() {
    byte error, address;
    int nDevices;

    Serial.println("Scanning...");

    nDevices = 0;
    for(address = 1; address < 127; address++) {
        // The i2c_scanner uses the return value of
        // the Wire.endTransmission() to see if
        // a device did acknowledge to the address.
        Wire.beginTransmission(address);
        error = Wire.endTransmission();

        if (error == 0) {
            Serial.print("I2C device found at address 0x");
            if (address < 16)
                Serial.print("0");
            Serial.println(address, HEX);
            if (address == I2C_DEFAULT){
                Wire.beginTransmission(address);
                Wire.write(0x6A);
                int data_t = Wire.endTransmission();
                Serial.print("sending data ");
                Serial.print(address);
                Serial.print(" result: ");
                Serial.println(data_t);
            }

            nDevices++;
        }
        else if (error == 4) {
            Serial.print("Unknown error at address 0x");
            if (address < 16)
                Serial.print("0");
            Serial.println(address, HEX);
        }
    }
    if (nDevices == 0)
        Serial.println("No I2C devices found");
    else
        Serial.println("done");
}

```

```

    delay(5000);    // wait 5 seconds for next scan
}

```

## A1.3 - I2C\_DynamicAddress\_Slave.ino

```

/*****
 *      Title: I2C_DynamicAddress_Slave
 *      Author: Jimmy Wong
 *      Date: Nov 18, 2018
 *      Code version: 0.0.1
 *****/

#include <Wire.h>

// Pin setup
#define BTN_PIN    PA5

int buttonState;
int lastButtonState = 0;
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;

// Size of each Tile matrix
const uint8_t matrixWidth = 4;
const uint8_t matrixHeight = 4;

// Number of Tile matrices
const uint8_t TilesX = 1;
const uint8_t TilesY = 1;

// I2C
uint8_t I2C_ADDR = 0x21; //Initialize I2C_ADDR
#define I2C_DEFAULT 0x42

void setup()
{
    //I2C Setup
    Wire.begin(I2C_DEFAULT); // join i2c bus with the default address
    Wire.onRequest(requestEvent); // register event
    Wire.onReceive(receiveEvent); // register event

    //Button Setup
    pinMode(BTN_PIN, INPUT);

    Serial.begin(9600); // start serial for output
}

uint8_t column = 0;

void loop()
{
    int reading = digitalRead(BTN_PIN);
    //buttonEvent
    if (reading != lastButtonState) {
        lastDebounceTime = millis();
    }
    if ((millis() - lastDebounceTime) > debounceDelay) {
        if (reading != buttonState){
            buttonState = reading;

```

```

        //Serial.print("button pressed but don't know value: ");
        //Serial.println(buttonState);
        if (buttonState == 1){
            //Turn off I2C and reinitialize
            Wire.end();
            delay(100); //slight delay before trying to reinitiali
            Wire.begin(I2C_ADDR); //join the i2c bus with a different add
            Wire.onRequest(requestEvent);
            Wire.onReceive(receiveEvent);
            Serial.print("Attempted Address Change");
            Serial.print(I2C_ADDR);
        }
    }
}

lastButtonState = reading;
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
    if (Wire.available()) //loop through all but the last
    {
        I2C_ADDR = Wire.read(); //receive byte as a int
    }
    Serial.println("received value");
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent()
{
    Wire.write("hello \n"); // respond with message of 6 bytes
    // as expected by master
}

```

## A1.4 - I2C\_DynamicDisplay\_Master.ino

```

/*****
 * Title: I2C_DynamicDisplay_Master
 * Author: Jimmy Wong
 * Date: Nov 18, 2018
 * Code version: 0.0.1
 *****/

#include <Wire.h>

#define I2C_ADDR 2

int x = 0;
byte LED_state = 0;
// Size of each Tile matrix
const uint8_t MATRIX_WIDTH = 4;
const uint8_t MATRIX_HEIGHT = 4;

#define LED1_PIN PA_1
#define LED2_PIN PA2
#define LED3_PIN PA3
#define LED4_PIN PA4
int LED_array [4] = {LED1_PIN, LED2_PIN, LED3_PIN, LED4_PIN};

```

```

unsigned int TilesX = 1;
void setup()
{
    Wire.begin();          // join i2c bus (address optional for master)
    pinMode(LED1_PIN, OUTPUT);
    pinMode(LED2_PIN, OUTPUT);
    pinMode(LED3_PIN, OUTPUT);
    pinMode(LED4_PIN, OUTPUT);
    Serial.begin(9600); // start serial for output
}

uint8_t column_max = 8;
uint8_t column = 0;
uint8_t Tile = 0;
uint8_t Tile_column;

void loop()
{
    //Check if the slave device exists on the bus
    Wire.beginTransmission(2);
    int error = Wire.endTransmission();
    if(error == 0) {
        Serial.print("Device connected at address 2");
        TilesX = 2;
    }else{
        TilesX = 1;
    }
    column_max = MATRIX_WIDTH * TilesX;
    column++;
    if(column >= column_max){
        column = 0;
    }
    Serial.print(" column: ");
    Serial.print(column);
    Tile_column = column % MATRIX_WIDTH;
    Tile = (column - Tile_column) / MATRIX_WIDTH;
    digitalWrite(LED1_PIN,LOW);
    digitalWrite(LED2_PIN,LOW);
    digitalWrite(LED3_PIN,LOW);
    digitalWrite(LED4_PIN,LOW);
    Serial.print(" Tile #: ");
    Serial.print(Tile);
    Serial.print(" Tile_column: ");
    Serial.println(Tile_column);
    switch(Tile){
        case 0:
            digitalWrite(LED_array[Tile_column],HIGH);
            if(TilesX == 2){
                Wire.beginTransmission(2);
                Wire.write(MATRIX_WIDTH+1);
                Wire.endTransmission();
            }
            break;
        case 1:
            Wire.beginTransmission(2);
            Wire.write(Tile_column);
            Wire.endTransmission();
            Serial.println("Data Sent");
            break;
        default:
    }
}

```

```

        break;
    }

    delay(100);
}

```

## A1.5 - I2C\_DynamicDisplay\_Slave.ino

```

/*****
 *      Title: I2C_Dyna   micDisplay_Slave
 *      Author: Jimmy Wong
 *      Date: Nov 18, 2018
 *      Code version: 0.0.1
 *****/

#include <Wire.h>

// Pin   setup
const uint8_t MATRIX_DATA_PIN = 4;
const uint8_t MATRIX_CLK_PIN = 7;
#define LED1_PIN PA1
#define LED2_PIN PA2
#define LED3_PIN PA3
#define LED4_PIN PA4
int LED_array [4] = {LED1_PIN, LED2_PIN, LED3_PIN, LED4_PIN};

// Size of each Tile   e matrix
const uint8_t MATRIX_WIDTH = 4;
const uint8_t MATRIX_HEIGHT = 4;

// Number of Tile matrices
const uint8_t TilesX = 1;
const uint8_t TilesY = 1;

#define I2C_ADDR 2
void setup()
{
    Wire.begin(I2C_ADDR);    // join i2c bus with address #2
    Wire.onRequest (requestEvent); // register event
    Wire.onReceive(receiveEvent); // register event

    pinMode(LED1_PIN, OUTPUT);
    pinMode(LED2_PIN, OUTPUT);
    pinMode(LED3_PIN, OUTPUT);
    pinMode(LED4_PIN, OUTPUT);

    Serial.begin(9600);    // start serial for      output
}

uint8_t column = MATRIX_WIDTH + 1;

void loop()
{
    digitalWrite(LED1_PIN, LOW);
    digitalWrite(LED2_PIN, LOW);
    digitalWrite(LED3_PIN, LOW);
    digitalWrite(LED4_PIN, LOW);
    if (column < MATRIX_WIDTH){
        digitalWrite(LED_array[column-1], HIGH);
    }
}

```

```

    }
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
    Serial.println("receiving data: ");
    if(Wire.available()){
        column = Wire.read() ; //receive by te as a int
        Serial.println(column);
    }
}

```

## A1.6 - I2C\_LED\_Master.ino

```

/*****
*      Title: I2C_LED_Master
*      Author: Jimmy Wong
*      Date: Nov 8, 2018
*      Code version: 0.0.1
*****/

#include <Wire.h>
#define I2C_ADDR 2
byte x = 0;
byte LED_state = 0;
void setup()
{
    Wire.begin(); // join i2c bus (address optional for master)
    Serial.begin(9600); // start serial for output
}
void loop()
{
    Wire.requestFrom(I2C_ADDR, 6); // request 6 bytes from slave device
    while(Wire.available()) // slave may send less than requested
    {
        char c = Wire.read(); // receive a byte as char
        Serial.print(c); // print the character
    }
    delay(10);
    LED_state = x % 2;
    Wire.beginTransmission(I2C_ADDR); // transmit to device
    Wire.write(LED_state); // sends one byte
    Wire.endTransmission(); // stop transmitting
    x++;
    delay(500);
}

```

## A1.7 - I2C\_LED\_Slave.ino

```

/*****
*      Title: I2C_LED_Slave
*      Author: Jimmy Wong
*      Date: Nov 8, 2018
*      Code version: 0.0.1
*****/

#include <Wire.h>
#define I2C_ADDR 4
const int ledPin = PC13;

```



```

void setup()
{
  Wire.begin(I2C_ADDR); // join i2c bus with address #2
  Wire.onRequest(requestEvent); // register event
  Wire.onReceive(receiveEvent); // register event
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600); // start serial for output
}

void loop()
{
  // function that executes whenever data is received from master
  // this function is registered as an event, see setup()
  void receiveEvent(int howMany)
  {
    int x = Wire.read(); // receive byte as an integer
    if(x == 1){
      digitalWrite(ledPin, HIGH); // print the integer
    }else{
      digitalWrite(ledPin, LOW);
    }
  }
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent()
{
  Wire.write("hello \n"); // respond with message of 6 bytes
                             // as expected by master
}

```

## A1.8 - IR\_MATRIX\_POLL.ino

```

/*****
 * Title: MatrixTestStm32
 * Author: Rowan Baker - French, Sanket Mittal
 * Date: Nov 22, 2018
 * Code version: 0.0.1
 * Availability:
 https://github.com/Rowansda bomb/T25ELEC491/blob/master/IRSensorsSTM32/IR_MATRIX_POLL.ino
 *****/

#include <Adafruit_GFX.h>
#include <Adafruit_DotStarMatrix.h>
#include <Adafruit_DotStar.h>

// Pin setup
const uint8_t MATRIX_DATA_PIN = PB10;
const uint8_t MATRIX_CLK_PIN = PB11;

// Size of each Tile matrix
const uint8_t matrixWidth = 4;
const uint8_t matrixHeight = 4;

// Number of Tile matrices
const uint8_t TilesX = 1;
const uint8_t TilesY = 1;

Adafruit_DotStarMatrix matrix = Adafruit_DotStarMatrix(
  matrixWidth,
  matrixHeight,
  TilesX,

```

```

    TilesY,
    MATRIX_DATA_PIN,
    MATRIX_CLK_PIN,
    DS_MATRIX_TOP + DS_MATRIX_LEFT +
    DS_MATRIX_COLUMNS + DS_MATRIX_ZIGZAG + DS_TILE_PROGRESSIVE,
    DOTSTAR_BGR
);

const uint16_t colors[] = {
    matrix.Color(255, 0, 0),          matrix.Color(0, 255, 0), matrix.Color(0, 0, 255), matrix.Color(255,
    255, 255), matrix.Color(0,0,0)
};

const uint8_t sensorDataSize = 16;
volatile int sensorData[sensorDataSize] = {0};

const uint8_t led = PC13;
const uint8_t testPin = PB12;

volatile bool interruptFlag = false;

HardwareTimer timer(2);

void setup() {

    #if defined(__AVR_ATtiny85__) && (F_CPU == 16000000L)
    clock_prescale_set(clock_div_1); // Enable 16 MHz on Tri
    #endif

    matrix.begin(); // Initialize pins for output
    matrix.setBrightness(64); // Set max brightness (out of 255)

    pinMode(led, OUTPUT);
    pinMode(testPin, OUTPUT);

    Serial.begin(38400);

    // set sensor polling interrupt routine
    // Pause the timer while we're configuring it
    timer.pause();

    // Set up period
    timer.setPeriod(SENSOR_POLL_PERIOD*1000); // in microseconds

    // Set up an interrupt on channel 1
    timer.setChannel1Mode(TIMER_OUTPUT_COMPARE);
    timer.setCompare(TIMER_CH1, 1); // Interrupt 1 count after each update
    timer.attachCompare1Interrupt(sensorRead);

    // Refresh the timer's count, prescale, and overflow
    timer.refresh();

    // Start the timer counting
    timer.resume();
}

void sensorRead() {
    uint8_t si = 0;
    uint8_t pin = 0;
    gpio_write_bit(GPIOB, 12, HIGH);

```

```

for(uint8_t i = (sensorDataSize - 1); i > 5; -- i) {
    pin = (sensorDataSize - 1) - i;
    if(pin < 8){
        sensorData[i] = analogRead(pin)/16;
    }
    else {
        sensorData[i] = analogRead(pin + 10)/16;
    }
}

gpio_write_bit(GPIOB, 12, LOW);
interruptFlag = true;
}

void printSensorData() {
    uint16_t color;
    matrix.setCursor(0, 0);

    for (uint8_t i = 0; i < matrixWidth; i++) {
        for(uint8_t j = 0; j < matrixHeight; j++) {
            Serial.print(sensorData[i + j*(matrixHeight)]);

            if(j < matrixHeight - 1)
                Serial.print(" | ");
            else
                Serial.println();
        }
        Serial.println();
    }
}

int Color_Brightness(int sensorData){
    uint16_t color = matrix.Color(sensorData, 0, 0);
    return color;
}

void LEDControl() {
    uint16_t color;
    int adjusted;
    for (uint8_t i = 0; i < matrixWidth; i++) {
        for(uint8_t j = 0; j < matrixHeight; j++) {
            adjusted = sensorData[i + j*(matrixHeight)];
            if(adjusted > 20)
                adjusted *= 8;
            if (adjusted > 255)
                adjusted = 255;
            color = Color_Brightness(adjusted);
            matrix.drawPixel(i, j, color);
            matrix.show();
        }
    }
}

//Simply a while loop that gets the sensor poll data and prints it on update
void loop() {
    if (interruptFlag){

        //print the sensor data
        printSensorData();

        LEDControl();
    }
}

```

```

        //reset flag
        interruptFlag = false;
    }
}

```

## A1.9 - IRSensorsSTM32.ino

```

/**** *****
*      Title: IRSensorsSTM32
*      Author: Rowan Baker    - French
*      Date: Nov 22, 2018
*      Code version: 0.0.1
*      Availability:
http://raw.githubusercontent.com/Rowansdabomb/T25ELEC491/master/IRsensorsSTM32/IRsensorsSTM32.ino
*****/

#define SENSOR_POLL_PERIOD 100

const uint8_t sensorDataSize = 16;
volatile int sensorData[sensorDataSize] = {0};

const uint8_t led = PC13;
const uint8_t testPin = PB12;

const uint8_t matrixWidth = 4;
const uint8_t matrixHeight = 4;

volatile bool interruptFlag = false;
volatile unsigned long interruptTime = 0;

HardwareTimer timer(2);

void setup() {
    // put your setup code here, to run once:
    pinMode(led, OUTPUT);
    pinMode(testPin, OUTPUT);
    Serial.begin(38400);
    // set sensor polling interrupt routine
    // Pause the timer while we're configuring it
    timer.pause();

    // Set up period
    timer.setPeriod(SENSOR_POLL_PERIOD*1000); // in microseconds

    // Set up an interrupt on channel 1
    timer.setChannel1Mode(TIMESTAMP_OUTPUT_COMPARE);
    timer.setCompare(TIMESTAMP_CH1, 1); // Interrupt 1 count after each update
    timer.attachCompare1Interrupt(sensorRead);

    // Refresh the timer's count, prescale, and overflow
    timer.refresh();

    // Start the timer counting
    timer.resume();
}

void sensorRead() {
    uint8_t si = 0;
    uint8_t pin = 0;
    gpio_write_bit(GPIOB, 12, HIGH);

```

```

for(uint8_t i = (sensorDataSize - 1); i > 5; -- i) {
    pin = (sensorDataSize - 1) - i;
    if(pin < 8){
        sensorData[i] = analogRead(pin)/16; //convert to 255
    }
    // sensorData[i] = pin;
    }
    else {
        sensorData[i] = analogRead(pin + 10)/16;
    }
    // sensorData[i] = pin + 10;
    }
}
gpio_write_bit(GPIOB, 12, LOW);
interruptFlag = true;
}

void printSensorData() {
    Serial.write(27); // ESC command
    Serial.print("[2J"); // clear screen command
    Serial.write(27);
    Serial.print("[H"); // cursor to home command

    for (uint8_t i = 0; i < matrixWidth; i++) {
        for (uint8_t j = 0; j < matrixHeight; j++) {
            Serial.print(sensorData[i + j*(matrixHeight)]);
            if(j < matrixHeight - 1)
                Serial.print(" | ");
            else
                Serial.println();
        }
    }
    Serial.println();
}

//Simply a while loop that gets the sensor poll data and prints it on update
void loop() {
    // put your main code here, to run repeatedly:

    Serial.println("Let's begin!");
    while(true){
        // digitalWrite(led, HIGH);
        // delay(200);
        // digitalWrite(led, LOW);
        // delay(200);

        if (interruptFlag){
            if (digitalRead(led) == HIGH) {
                digitalWrite(led, LOW);
            } else {
                digitalWrite(led, HIGH);
            }
        }

        //print the sensor data
        printSensorData();

        //reset flag
        interruptFlag = false;
    }
}

```

## A1.10 - MatrixTestStm32.ino

```
/*
 * Title: MatrixTestStm32
 * Author: Rowan Baker - French, Jimmy Wong
 * Date: Nov 10, 2018
 * Code version: 0.0.1
 * Availability:
 https://github.com/Rowansdabomb/T25ELEC491/edit/master/MatrixTestStm32/MatrixTestStm32.ino
 */

#include <Adafruit_GFX.h>
#include <Adafruit_DotStarMatrix.h>
#include <Adafruit_DotStar.h>

// Pin setup
const uint8_t MATRIX_DATA_PIN = PB10;
const uint8_t MATRIX_CLK_PIN = PB11;
const uint8_t TEST_PIN = PA7;

const uint8_t CHAR_WIDTH = 5;
const uint8_t CHAR_HEIGHT = 8;

// Size of each Tile matrix
const uint8_t matrixWidth = 4;
const uint8_t matrixHeight = 4;

// Number of Tile matrices
const uint8_t TilesX = 1;
const uint8_t TilesY = 1;

// Last argument: line 30 in Adafruit_DotStar.h for mappings
Adafruit_DotStarMatrix matrix = Adafruit_DotStarMatrix(
  matrixWidth,
  matrixHeight,
  TilesX,
  TilesY,
  MATRIX_DATA_PIN,
  MATRIX_CLK_PIN,
  DS_MATRIX_TOP + DS_MATRIX_LEFT +
  DS_MATRIX_COLUMNS + DS_MATRIX_ZIGZAG + DS_TILE_PROGRESSIVE,
  DOTSTAR_RGB
);

// See https://
const uint16_t colors[] = {
  matrix.Color(255, 0, 0), matrix.Color(0, 255, 0), matrix.Color(0, 0, 255), matrix.Color(255,
  255, 255)
};

void setup() {
  pinMode(TEST_PIN, OUTPUT);

  matrix.begin(); // Initialize pins for output

  matrix.setBrightness(64); // Set max brightness (out of 255)

  matrix.setTextWrap(false);
  matrix.setTextColor(colors[0]);
  matrix.show(); // Turn all LEDs off ASAP
}
```

```

    Serial.begin(9600);
}

int x = matrix.width();
int pass = 0;
uint8_t p_x = 0;
uint8_t p_y = 0;
uint8_t colorIndex = 0;

void scrollText(uint8_t fps, char* text, int t textLength) {
    // Test Conditions:
    // variables: matrixWidth, matrixHeight,
    // No delay

    // Results: (same for text = "3 2 1" and text = "6 5 4 3 2 1")
    // Max frame rate
    // 16x16 = 57fps
    // 4x4 = 96fps
    // SetupPeriod
    // 16x16 = 17.5ms
    // 4x4 < 1.25ms

    // Takes in an fps and text to scroll
    gpio_write_bit(GPIOA, 7, HIGH);
    matrix.fillScreen(0);
    matrix.setCursor(x, 0);
    matrix.print(F(text));
    if( -- x < - textLength*CHAR_WIDTH) {
        x = matrix.width();
    }
    matrix.show();
    gpio_write_bit(GPIOA, 7, LOW);
    delay(1000/fps);
}

void testEachPixel(uint8_t fps) {
    matrix.fillScreen(0);
    matrix.setCursor(x, 0);
    uint16_t color = colors[3];
    Serial.println(color);
    matrix.drawPixel(p_x, p_y, color);
    p_x++;
    if (p_x >= matrixWidth) {
        p_x = 0;
        p_y++;
    }
    if (p_y >= matrixHeight) {
        p_y = 0;
    }
    matrix.show();
    delay(1000/fps);
}

uint8_t i = 16;

void testAllPixels(uint8_t fps) {
    matrix.setCursor(0, 0);
    matrix.fillRect(0, 0, matrixWidth, 4*matrixHeight, colors[1]);

    Serial.println(colorIndex);
}

```

```

    colorIndex++;
    if (colorIndex >= 4){
        colorIndex = 0;
        matrix.setBrightness(255/(i*2));
        Serial.print("Color Brightness ");
        Serial.println(256/i);
        i*=2;
        if(i > 16)
            i = 1;
    }

    matrix.show();
    delay(1000/fps);
}

void testDigital() {
    // produces 672.4kHz square wave
    // risetime (3V) < 42ns
    // falltime (.3V) < 47ns
    digitalWrite(PA7, HIGH);
    digitalWrite(PA7, LOW);
}

void testGPIO() {
    // produces 1.8 93MHz pulse wave
    // risetime (2.5V) < 18ns (rises only to 2.7V)
    // falltime (.32V) < 52ns

    gpio_write_bit(GPIOA, 7, HIGH);
    gpio_write_bit(GPIOA, 7, LOW);
}

void loop() {
    char text[] = "~ * ^";

    scrollText(10, text, sizeof(text)/sizeof(text[0]));

    // testDigital();
    // testGPIO();

    // testEachPixel(8);
    // testAllPixels(1);
}

```

## A1.11 - TopologyTest\_Master.ino

```

/*****
* Title: Topology Test Master
* Author: Jimmy Wong
* Date: February 10, 2019
* Code version: 0.0.3
*****/

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_DotStarMatrix.h>
#include <Adafruit_DotStar.h>

#define I2C_DEFAULT 0x42

```



```

#define CNCT_U B0001
#define CNCT_D B0010
#define CNCT_L B0100
#define CNCT_R B1000

#define PIN_DIR_U PA4
#define PIN_DIR_D PA6
#define PIN_DIR_L PA3
#define PIN_DIR_R PA5

#define TILE_MAX 5

#define DEBUG 0

int print_flag = 0;
#define PRINT_EN 1

#define ARRAY_SIZE 7

//Use layout to store the addresses of the devices 7 by 7
int layout[7][7] = {{ 0, 0, 0, 0, 0, 0, 0},
                    { 0, 0, 0, 0, 0, 0, 0},
                    { 0, 0, 0, 0, 0, 0, 0},
                    { 0, 0, 0, 9, 0, 0, 0},
                    { 0, 0, 0, 0, 0, 0, 0},
                    { 0, 0, 0, 0, 0, 0, 0},
                    { 0, 0, 0, 0, 0, 0, 0}},

};

int Tile_order[4] = {0, 0, 0, 0};

struct POS {
    int x;
    int y;
};

/* Tile structure
 * active = is this Tile currently active
 * addr = address of the current Tile
 * posX = position of Tile in the X direction
 * posY = position of Tile in the Y direction
 * ports = state of the directional pins
 * ports_pre = state of the directional pins in the previous instance
 */
struct TILE {
    byte active;
    int addr;
    POS pos;
    int ports;
    int ports_pre;
};

TILE Tile[TILE_MAX];

//last set as default
int addr_list[TILE_MAX] = {0x08, 0x10, 0x18, 0x20, 0x28};

//int error;
int i;

```

```

int TileID;
int x_free;
int y_free;
int dirChange;
int dirChange_f;
int Tile_count;
int Tile_order_f = 0;
//int Tile_count_pre;
int show_Tile;
int show_x;
int show_y;

int pinDir = B00000000;

//DotStar Setup
const uint8_t MATRIX_DATA_PIN = PB11;
const uint8_t MATRIX_CLK_PIN = PB10;
const uint8_t CHAR_WIDTH = 5;
const uint8_t CHAR_HEIGHT = 8;

// Size of each Tile matrix
const uint8_t matrixWidth = 4;
const uint8_t matrixHeight = 4;

// Number of Tile matrices
const uint8_t TilesX = 1;
const uint8_t TilesY = 1;

Adafruit_DotStarMatrix matrix = Adafruit_DotStarMatrix(
    matrixWidth,
    matrixHeight,
    TilesX,
    TilesY,
    MATRIX_DATA_PIN,
    MATRIX_CLK_PIN,
    DS_MATRIX_TOP + DS_MATRIX_LEFT +
    DS_MATRIX_COLUMNS + DS_MATRIX_ZIGZAG + DS_TILE_PROGRESSIVE,
    DOTSTAR_RGB
);

const uint16_t colors[] = {
    matrix.Color(255, 0, 0), matrix.Color(0, 255, 0), matrix.Color(0, 0, 255), matrix.Color(255,
    255, 255)
};

void handler_tile_m(void);

void show_tile_info(int TileID);

void setup() {
    // Directional Pin Setup

    pinMode(PIN_DIR_U, INPUT_PULLDOWN);
    pinMode(PIN_DIR_D, INPUT_PULLDOWN);
    pinMode(PIN_DIR_L, INPUT_PULLDOWN);
    pinMode(PIN_DIR_R, INPUT_PULLDOWN);

    // Internal Device Map - Initial Population
    for(i = 0; i < TILE_MAX; i++){
        Tile[i].active = 0;
        Tile[i].addr = addr_list[i];
    }
}

```

```

    Tile[i].pos.x = 0;
    Tile[i].pos.y = 0;
    Tile[i].ports = B00000000;
}

Tile[0].active = 1;
Tile[0].addr = 0xFF;
Tile[0].pos.x = 3;
Tile[0].pos.y = 3;

//Timer for testing purposes
Timer2.setMode(TIMER_CH1, TIMER_OUTPUTCOMPARE);
Timer2.setPeriod(1000000);
Timer2.setCompare(TIMER_CH1, 1);
Timer2.attachInterrupt(TIMER_CH1, handler_tim);

// I2C Master Setup
Wire.begin();

// DotStar Setup
matrix.begin(); // Initialize pins for output
matrix.setBrightness(64); // Set max brightness (out of 255)
matrix.setTextWrap(false);
matrix.setTextColor(colors[0]);
matrix.show(); // Turn all LEDs off ASAP

// Serial Setup - for output
Serial.begin(9600);

Tile_order_f = 1;
}

int x = matrix.width();
int pass = 0;
uint8_t colorIndex = 0;

void loop() {

    if(print_flag == PRINT_EN){

        Serial.print("x free: ");
        Serial.print(x_free);
        Serial.print(", y_free: ");
        Serial.println(y_free);

        show_Tile_info(0);
        show_Tile_info(1);
        show_Tile_info(2);
        show_Tile_info(3);
        show_Tile_info(4);

        Serial.print("# Tiles: ");
        Serial.print(Tile_count);
        Serial.print(" Order: ");
        for(int j = 0; j < 4; j++){
            Serial.print(Tile_order[j]);
            Serial.print(" ");
        }
        Serial.println();
    }
}

```

```

    //Disable the flag
    //print_flag = 0;
}

int array_x_max = 3;
int array_y_max = 3;
int array_x_min = 3;
int array_y_min = 3;

//Serial.println("Determining Directions");

//Determine occupied directions
Tile[0].ports_pre = Tile[0].ports;
Tile[0].ports = B0000;
if(digitalRead(PIN_DIR_U)){
    Tile[0].ports = Tile[0].ports | CNCT_U;
}
if(digitalRead(PIN_DIR_D)){
    Tile[0].ports = Tile[0].ports | CNCT_D;
}
if(digitalRead(PIN_DIR_L)){
    Tile[0].ports = Tile[0].ports | CNCT_L;
}
if(digitalRead(PIN_DIR_R)){
    Tile[0].ports = Tile[0].ports | CNCT_R;
}

// Loop to check if the currently existing Tiles
// still exist, if not clear and erase from layout
// Tile[0] will be reserved for the master
//Tile_count_pre = Tile_count;
Tile_count = 1;

//Serial.println("First I2C Check");
Wire.beginTransmission(I2C_DEFAULT);
int chk_error = Wire.endTransmission();
if (chk_error == 0){
    Serial.println("Default Address still detected");
}

for(i = 0; i < TILE_MAX; i++){
    int error;
    //Serial.print("Currently Checking Tile ");
    //Serial.println(i);
    if( i != 0 ){// Check if dealing with master Tile
        if( Tile[i].active == 1 ){
            Wire.beginTransmission(Tile[i].addr);
            error = Wire.endTransmission();
        }else{
            TileID = i;
        }

        if (error == SUCCESS) {
            if(DEBUG){
                Serial.print("I2C device found at address 0x");
                Serial.println(Tile[i].addr, HEX);
            }//END DEBUG PRINT
            Tile[i].ports_pre = Tile[i].ports;
            //If available request current port status from slave devices
            Wire.requestFrom(Tile[i].addr, 1);

```

```

    Tile[i].ports = Wire.read();
    Tile_count++;
    if( Tile[i].pos.x < array_x_min ){
        array_x_min = Tile[i].pos.x;
    }
    if( Tile[i].pos.x > array_x_max ){
        array_x_max = Tile[i].pos.x;
    }
    if( Tile[i].pos.y > array_y_max ){
        array_y_max = Tile[i].pos.y;
    }
    if( Tile[i].pos.y < array_y_min ){
        array_y_min = Tile[i].pos.y;
    }
}
}else{
    if(DEBUG){
        Serial.print("No I2C device found at address 0x");
        Serial.println(Tile[i].addr, HEX);
    }
    }// END DEBUG PRINT
    layout[Tile[i].pos.y][Tile[i].pos.x] = 0;
    Tile[i].pos.x = 0;
    Tile[i].pos.y = 0;
    Tile[i].active = 0;
    //Tile Removed
    Tile_order_f = 1;
}
}

//Check if the directional ports has changed
if(Tile[i].ports != Tile[i].ports_pre){
    dirChange_f = 1;
    dirChange = Tile[i].ports ^ Tile[i].ports_pre;
    switch(dirChange){
        case CNCT_U:
            x_free = Tile[i].pos.x;
            y_free = Tile[i].pos.y - 1;
            break;
        case CNCT_D:
            x_free = Tile[i].pos.x;
            y_free = Tile[i].pos.y + 1;
            break;
        case CNCT_L:
            x_free = Tile[i].pos.x - 1;
            y_free = Tile[i].pos.y;
            break;
        case CNCT_R:
            x_free = Tile[i].pos.x + 1;
            y_free = Tile[i].pos.y;
            break;
        default:
            //will not happen
            break;
    }
}
}

}

//End of Directional ports changing
}

}

if(dirChange_f == 1 ){

    // Check if the default address exist

```

```

Wire.beginTransmission(I2C_DEFAULT);
int def_error = Wire.endTransmission();
//Serial.println(def_error);
if (def_error == SUCCESS){
    Serial.println("Device found at default address");
    Wire.beginTransmission(I2C_DEFAULT);
    Wire.write('A');
    Wire.write(Tile[TileID].addr); //Assign the next available address from
    Wire.endTransmission();
    //Maybe insert something here during the connection process?
    Serial.print("Sent address: ");
    Serial.println(Tile[TileID].addr, HEX);
}

delay(500); //Half second delay before checking that the Tile is now in place
//Determine the location of the Tile
Wire.beginTransmission(Tile[TileID].addr);
int addr_error = Wire.endTransmission();
if (addr_error == 0){
    Tile[TileID].active = 1;
    Tile[TileID].pos.x = x_free;
    Tile[TileID].pos.y = y_free;
    layout[y_free][x_free] = TileID;
    x_free = 0;
    y_free = 0;
    dirChange_f = 0; // reset the direction changed flag
    Tile_order_f = 1; // raise the flag for to redo the Tile order
}

}
if (print_flag == PRINT_EN){
    Serial.println("Current Internal Array");
    for(int j = 0; j < ARRAY_SIZE; j++){
        for(int k = 0; k < ARRAY_SIZE; k++){
            Serial.print(layout[j][k]);
            Serial.print(" ");
        }
        Serial.println();
    }
    print_flag = 0;
}
if(print_flag == 5){
    Serial.print("Array X Values: ");
    Serial.print(array_x_min);
    Serial.print(" ");
    Serial.println(array_x_max);
    Serial.print("Array Y Values: ");
    Serial.print(array_y_min);
    Serial.print(" ");
    Serial.println(array_y_max);
    print_flag = 0;
}

//TODO: Sending data dynamically
if(Tile_order_f == 1){ // only needs to be done if number of Tiles changes
    int cnt_x;
    int cnt_y;
    int cnt_order = 0;
    for(cnt_order = 0; cnt_order < 4; cnt_order++){
        Tile_order[cnt_order] = 0; //reset the order
    }
}

```

```

    }
    cnt_order = 0;
    for(cnt_y = array_y_min; cnt_y <= array_y_max; cnt_y++){
        for(cnt_x = array_x_min; cnt_x <= array_x_max; cnt_x++){
            /*Serial.print(cnt_x);
            Serial.print("At position ");
            Serial.print(cnt_x);
            Serial.print(" ");
            Serial.println(cnt_y);
            */
            int temp_id = layout[cnt_y][cnt_x];
            if (temp_id == 9){
                Tile_order[cnt_order] = temp_id;
                cnt_order++;
            }else if (temp_id != 0){
                if( Tile[temp_id].active == 1){
                    Tile_order[cnt_order] = temp_id;
                    cnt_order++;
                }
            }
        }
        Tile_order_f = 0;
    } // End looping through array
} // END if

//Tile_order[x] is the index of the Tile
for(show_Tile = 0; show_Tile < Tile_count; show_Tile++){

    if(Tile_order[show_Tile] == 9){
        matrix.fillScreen(0);
        if((Tile[0].ports & CNCT_U) == CNCT_U){
            matrix.fillRect(1, 3, 2, 1, colors[show_Tile]);
        }
        if((Tile[0].ports & CNCT_D) == CNCT_D){
            matrix.fillRect(1, 0, 2, 1, colors[show_Tile]);
        }
        if((Tile[0].ports & CNCT_L) == CNCT_L){
            matrix.fillRect(0, 1, 1, 2, colors[show_Tile]);
        }
        if((Tile[0].ports & CNCT_R) == CNCT_R){
            matrix.fillRect(3, 1, 1, 2, colors[show_Tile]);
        }
        matrix.fillRect(1, 1, 2, 2, colors[3]);
        matrix.show();
    }else{
        Wire.beginTransmission(Tile[Tile_order[show_Tile]].addr);
        Wire.write('B');
        Wire.write(show_Tile);
        Wire.endTransmission();
    }
}
Serial.println("Reached the end");

delay(100);

}

void handler_tim(void) {
    print_flag = 1;
}

void show_Tile_info(int TileID){

```

```

Serial.print("Tile ID: ");
Serial.println(TileID);
Serial.print(" x: ");
Serial.print(Tile[TileID].pos.x);
Serial.print(", y: ");
Serial.print(Tile[TileID].pos.y);
Serial.print(", active: ");
Serial.print(Tile[TileID].active);
Serial.print(", ports: ");
Serial.println(Tile[TileID].ports, BIN);
}

```

## A1.12 - TopologyTest\_Slave.ino

```

/*****
* Title: Topology Test Slave
* Author: Jimmy Wong
* Date: February 10, 2019
* Code version: 0.0.3
*****/

#include <Wire_slave.h>
#include <Adafruit_GFX.h>
#include <Adafruit_DotStarMatrix.h>
#include <Adafruit_DotStar.h>

// Pin setup

#define CNCT_U B0001
#define CNCT_D B0010
#define CNCT_L B0100
#define CNCT_R B1000

#define PIN_DIR_U PA4
#define PIN_DIR_D PA6
#define PIN_DIR_L PA3
#define PIN_DIR_R PA5

#define LED_U PA3
#define LED_D PA4
#define LED_L PA5
#define LED_R PA2

//DotStar Setup
const uint8_t MATRIX_D ATA_PIN = PB11;
const uint8_t MATRIX_CLK_PIN = PB10;
const uint8_t CHAR_WIDTH = 5;
const uint8_t CHAR_HEIGHT = 8;

// Size of each Tile matrix
const uint8_t matrixWidth = 4;
const uint8_t matrixHeight = 4;

const uint8_t TilesX = 1;
const uint8_t TilesY = 1;

Adafruit_DotStarMatrix x matrix = Adafruit_DotStarMatrix(
  matrixWidth,

```



```

matrixHeight,
TilesX,
TilesY,
MATRIX_DATA_PIN,
MATRIX_CLK_PIN,
DS_MATRIX_TOP + DS_MATRIX_LEFT +
DS_MATRIX_COLUMNS + DS_MATRIX_ZIGZAG + DS_TILE_PROGRESSIVE,
DOTSTAR_BB
);

const uint16_t colors[] = {
    matrix.Color(255, 0, 0), matrix.Color(0, 255, 0), matrix.Color(0, 0, 255), matrix.Color(255,
255, 255)
};

int buttonState = 0;
// I2C
uint8_t I2C_ADDR = 0x42; //Initialize I2C_ADDR
#define I2C_DEFAULT 0x42

int pos_x = 0;
int pos_y = 0;

int dspy_en = 0;
int mtrx_en = 0;
int brightness = 0;

int led_out[2][2] = { {LED_L, LED_U},
                      {LED_D, LED_R}};

void handler_tim(void);

void setup()
{
    //I2C Setup
    Wire.begin(I2C_DEFAULT); // join i2c bus with the default address
    Wire.onRequest(requestEvent); // register event
    Wire.onReceive(receiveEvent); // register event

    // Directional Pin Setup
    pinMode(PIN_DIR_U, INPUT_PULLDOWN);
    pinMode(PIN_DIR_D, INPUT_PULLDOWN);
    pinMode(PIN_DIR_L, INPUT_PULLDOWN);
    pinMode(PIN_DIR_R, INPUT_PULLDOWN);
    /*
    // Directional LED setup **Replace with DotStar**
    pinMode(LED_U, OUTPUT);
    pinMode(LED_D, OUTPUT);
    pinMode(LED_L, OUTPUT);
    pinMode(LED_R, OUTPUT);*/

    // Timer Setup
    Timer2.setMode(TIMER_CH1, TIMER_OUTPUTCOMPARE);
    Timer2.setPeriod(1000000);
    Timer2.setCompare(TIMER_CH1, 1);
    Timer2.attachInterrupt(TIMER_CH1, handler_tim);

    // DotStar Setup - BEGIN
    matrix.begin();
    matrix.setBrightness(64); // Set max brightness
    matrix.setTextWrap(false);

```

```

matrix.setTextColor(colors[0]);
matrix.show(); // Turn all LEDs off ASAP

// Serial Setup      - for output
Serial.begin(9600);
}

int ports = B0000;
int temp_ports = B0000;
uint8_t column = 0;
char msg_ buf[6] = {0, 0,      0, 0, 0, 0};

void loop()
{
  //Serial.println(buttonState);
  if (buttonState == 1){
    //Turn off I2C and reinitialize
    Wire.begin(I2C_ADDR); //Join the i2c bus with a different address
    Wire.onRequest(requestEvent);
    Wire.onReceive(receiveEvent);
    Serial.print("Attempted Address Change");
    Serial.println(I2C_ADDR, HEX);
    buttonState = 2;
  }
  //Determine occupied directions
  temp_ports = B0000;
  if(digitalRead(PIN_DIR_U)){
    temp_ports = temp_ports | CNCT_U;
  }
  if(digitalRead(PIN_DIR_D)){
    temp_ports = temp_ports | CNCT_D;
  }
  if(digitalRead(PIN_DIR_L)){
    temp_ports = temp_ports | CNCT_L;
  }
  if(digitalRead(PIN_DIR_R)){
    temp_ports = temp_ports | CNCT_R;
  }
  ports = temp_ports;

  //Display
  //x position

  if(dspy_en){
    pos_x = (int) msg_buf[0] - 48;
    pos_y = (int) msg_buf[1] - 48;
    brightness = (int) msg_buf[2] * 25;
    digitalWrite(led_out[pos_x][pos_y], HIGH);
  }
  if(mtrx_en){
    Serial.print("Received: ");
    Serial.println(msg_buf[0]);
    matrix.fillScreen(0);
    if((ports & CNCT_U) == CNCT_U){
      matrix.fillRect(1, 3, 2, 1, colors[msg_buf[0]]);
    }
    if((ports & CNCT_D) == CNCT_D){
      matrix.fillRect(1, 0, 2, 1, colors[msg_buf[0]]);
    }
  }
}

```

```

    if((ports & CNCT_L) == CNCT_L){
        matrix.fillRect(0, 1, 1, 2, colors[msg_buf[0]]);
    }
    if((ports & CNCT_R) == CNCT_R){
        matrix.fillRect(3, 1, 1, 2, colors[msg_buf[0]]);
    }
    matrix.fillRect( 1, 1, 2, 2, colors[2]);
    matrix.show();
}

delay(100);
//Serial.println("continuing");
}

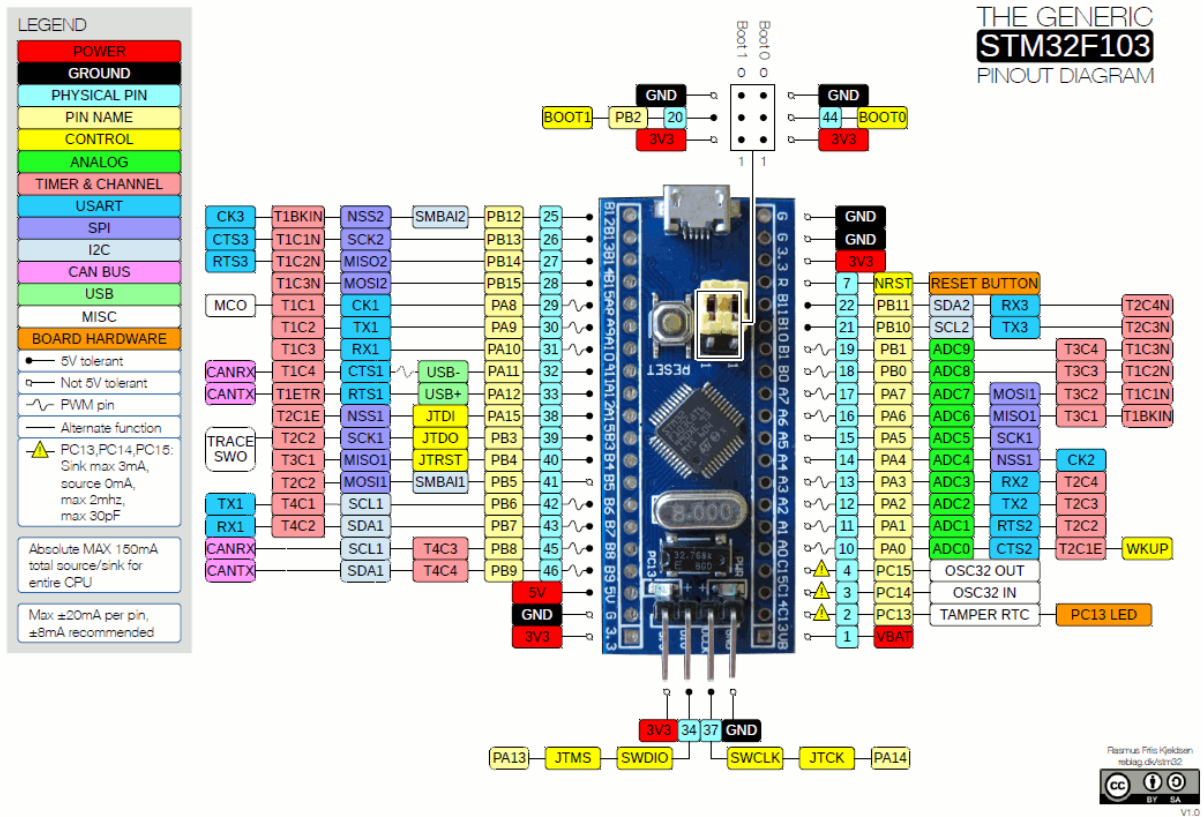
// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
    char c;
    Serial.println("Event Received");
    if(Wire.available()){
        c = Wire.read();    // receive first byte as a character
        Serial.println(c);    // print the character
    }
    if(c == 'A'){
        I2C_ADDR = Wire.read();    // receive byte as an integer
        Serial.println(I2C_ADDR, HEX);    // print the integ
        if(buttonState != 2){
            Wire.end();
            buttonState = 1;
        }
    }
    else if (c == 'E'){
        dspy_en = 1;
        int i = 0;
        while(Wire.available()){
            msg_buf[i] = Wire.read();
            Serial.print(msg_buf[i]);
            i++;
        }
        Serial.println();
    }
    else if (c == 'D'){
        dspy_en = 0;
        mtrx_en = 0;
    }
    else if (c == 'B'){
        mtrx_en = 1;
        int i = 0;
        while(Wire.available()){
            msg_buf[i] = Wire.read();
            Serial.print(msg_buf[i]);
            i++;
        }
    }
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent()
{
    Wire.write(ports); // respond with message of 1 byte
                        // as expected by master
}

```

```
}  
void handler_t im(void){  
    digitalWrite(led_out[0][0], LOW);  
    digitalWrite(led_out[1][0], LOW);  
    digitalWrite(led_out[0][1], LOW);  
    digitalWrite(led_out[1][1], LOW);  
}
```

## Appendix II: Pinout Diagrams



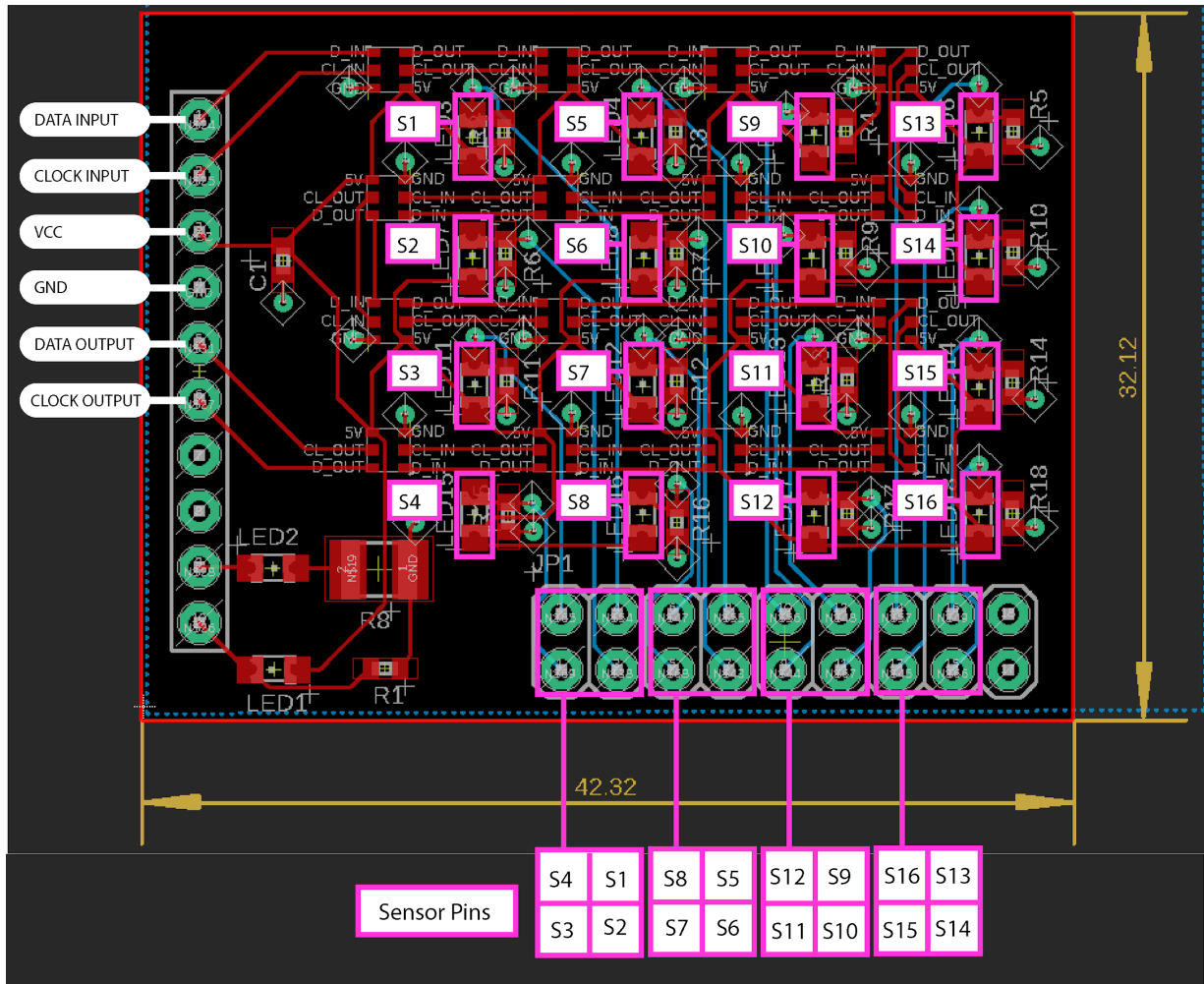


Fig 2: Pinout PCB Schematic

# Citations

- [1] OSRAM, "High Power Infrared Emitter (850 nm) Version 1.6", SFH 4056 datasheet, August. 2016
- [2] NIST, "Reflectance Measurements of Human Skin", National Institute of Standards and Technology, 2015 [Online] Available: <https://www.nist.gov>. [Accessed: Nov 21, 2018].
- [3] Vishay Semiconductors, "Silicon NPN Phototransistor", VEMT2000X01, VEMT2020X01 datasheet, August. 2011
- [4] I. Lee, S. Cho and G. Moon, "Interleaved Buck Converter Having Low Switching Losses and Improved Step-Down Conversion Ratio", IEEE Transactions on Power Electronics, vol. 27, no. 8, pp. 3664-3675, 2012.
- [5] "Chapter 5. The Discontinuous Conduction Mode", Ecee.colorado.edu, 2018. [Online]. Available: [http://ecee.colorado.edu/~ecen5797/course\\_material/Ch5slides.pdf](http://ecee.colorado.edu/~ecen5797/course_material/Ch5slides.pdf). [Accessed: 26- Nov- 2018].