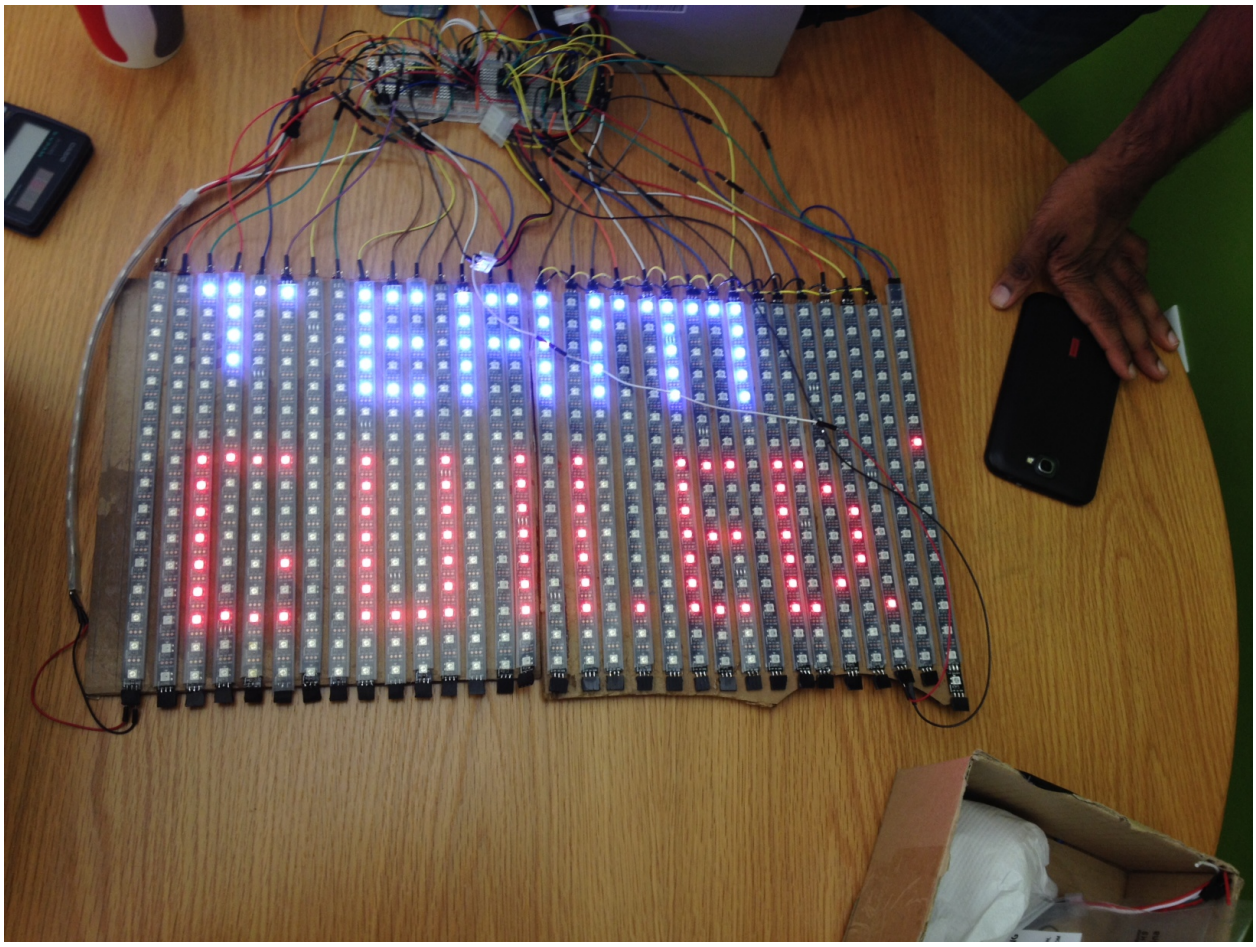
Final Project Report - EE 249

Introduction to Embedded Systems



By –

Phillip Azar

Peadar Keegan

Antonio de Lima Fernandes

Adarsh Mani

Modular LED Matrices: A proof of scalable, robust, and model-based design practices in low volume embedded systems projects.

Phillip Azar, Peadar Keegan, Adarsh Mani, Antonio De Lima Fernandes

1. Introduction

As we stated in our Project Charter, the goal of our project was to "create a configurable, modular, scalable and model based framework for the RGB LED Matrix Display using the WS2812 protocol". The aim of the project was to use model-based design to create something that can be scaled to an (almost) arbitrary size. We decided that a fundamental problem with many embedded systems projects is that they are often made to fit the original scale of the solution, and do not perform well if you try to make them larger or smaller. We propose a prototype that demonstrates a low-volume solution which incorporates model-based design principles for scalability and modularity.

2. Feasibility & Processing GUI

The major interactive portion of the project was the graphic user interface that was developed using the Java Processing tool. Processing is a multi-platform development suite that allows for the rapid usage and integration of a variety of graphical user interface and mathematical utility libraries. To effectively demonstrate the model-based design aspect of the project, an intuitive and easy to use GUI that incorporated a feasibility analyzer was a must. A diagram of this GUI is shown in Figure 2.1 below, illustrating its connection to various components of the system.

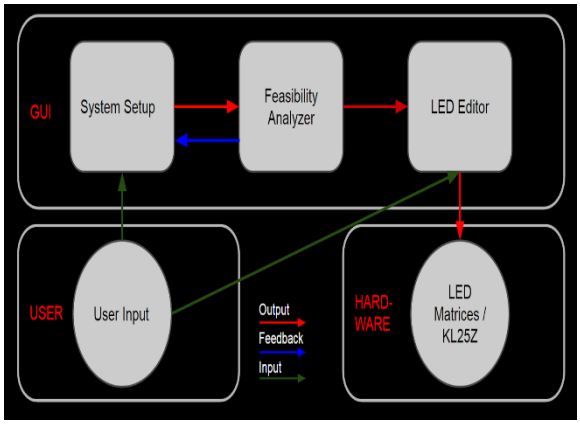


Figure 2.1 – High level system diagram of GUI.

While the user is drawing or uploading images to his or her LED workspace, the feasibility analyzer checks each pixel set and each image chosen to determine if it can be displayed or represented by the given workspace. Then, one of two things could occur: the analyzer returns an "all clear" signal to the GUI, and the user would continue drawing uninterrupted, or the analyzer would return a "failed requirements" signal. Upon receiving the "failed requirements signal," we pondered what we would have the GUI do to handle the situation. From a UI/UX perspective, we knew that the best possible experience would come from the uninterrupted and seamless use of the software. However, from a design perspective, leaving the user in the dark of potential design flaws in an area such as their electrical setup could be a costly mistake. We needed to strike a rich balance between helpful and informative, and so we decided that it would be best that all the insights provided by the simulator would be non-blocking until there was a critical flaw detected. Critical flaws included inadequate power supply for a board (under 500 mA in the case of a KL25Z), or unrealistic workspace setups (i.e. infinitely many columns of modules). In the event of a critical flaw, the program would not continue until the user changed his or her setup.

The feasibility analyzer essentially demonstrates how physical dynamics (or in this case, electrical dynamics) play a key role in maintaining the integrity of our system. Since the user is dealing with modules that collectively can source a large amount of current, micro-controllers can be very easily damaged or destroyed by inadequate setups. Our feasibility analyzer seeks to prevent this from occurring by calculating the electrical parameters such as current draw per LED depending on brightness.

3. Scalability & Modularity

To incorporate modularity and scalability, we wanted to allow a user to simply plug-and-play with modules with their setup. This meant allowing the user to connect any combination of blocks horizontally or vertically without changing any code. To facilitate this we needed hardware and software that could scale correctly. For horizontal connections, this was a straightforward problem to solve. The WS2811

library we used could send data to any number of LED's on a single strip, so there were no real software challenges involved in adding more blocks horizontally. The only problem lay in making hardware that could plug in to the existing blocks, which was simply a case of soldering female headers on to the end of our blocks to connect with the male headers of the next blocks.

Vertical connections, on the other hand, posed a significant technical challenge. We ultimately came up with two solutions for this: using demultiplexers (demux) to power many strips from the same Data Pin, and using SPI to communicate between two KL25z boards, each of which powered different blocks of LEDs.

The idea behind demultiplexers was to remove the reliance on a separate digital IO pin for each strip. We defined one basic block to be an 8x8 square of LEDs, so the goal was to light a block from a single pin. Figure 3.1 below shows the setup for the demultiplexer array.

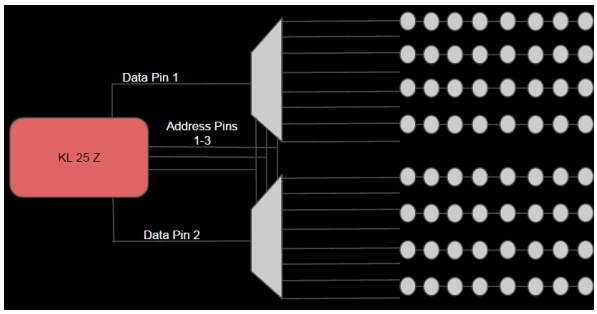


Figure 3.1 – Demultiplexer setup using KL25Z

Demultiplexing significantly reduces the number of IO pins needed, allowing many more blocks to be connected from a single microcontroller. Another advantage was that the output of the demux chips was proportional to their supply voltage, so this also solved the problem of providing a 5V input to the LED's from a 3.3V Digital Out on the KL25Z.

The software running on the KL25Z board listens on the serial port for data from the Processing GUI. Processing sends a standard packet with the RGB colors of the square (representing the LEDs in the workspace) the user clicked, along with the X and Y co-ordinates. It uses the co-ordinates to store the RGB bytes in the correct place in the board's memory, then uses the information about the Y co-ordinate to set the address pins of the multiplexers. The board then refreshes each data pin with the RGB sequence appropriate for the row being driven, which lights up the new LED and refreshes the blocks where nothing was changed.

The reason that the board refreshes every data pin, and not just the block to which a change was made, is twofold. First, it prevents any uncertainty as to what the output at the other data pins is. We encountered problems with unpredictable behavior before we implemented where certain pins would light up at random locations, with no real reason as to why this occurred. Second, it provides an upper limit on the amount of time each row goes without being refreshed. This is useful, for example, when you are uploading a full image and you want the image to appear instantaneously on the LED's.

There are three main challenges with our demultiplexing algorithm. First, it is not arbitrarily scalable. The KL25z must know how many and which data pins to initialize, which in turn means the programmer must hard code an upper limit into the software. If a larger number of vertical blocks is required the user must use a second microcontroller. Secondly, the demux must be given time to settle, especially when the address pins change. We tackled this quirk by changing the address pins at the earliest opportunity, but not starting the DMA until all other computations had finished. This introduced enough delay to ensure reliability. Thirdly, the software on the computer and the board kept separate copies of what the matrix should look like. This lead to the possibility of 'stale' data, with the GUI not reflecting what was being shown on the LEDs. This was done to minimize the amount of time on the serial port, while at the same time allowing the board to refresh the LEDs when it needed to. Perhaps with faster serial communication it would be feasible to keep a single data copy on the computer and send the relevant data every time an update was needed.

4. SPI communication

The second way of solving the scalability problem is to use multiple microcontrollers and establish communication between them. One microcontroller will act as

the master and initiate all communication with the other slave microcontrollers. It is also the responsibility of the master to communicate with the computer to get the user input with regards to which image he/she would like displayed on the LED matrix.

The arrangement of the different microcontrollers and the computer is set up as shown in figure 4.1 below. It follows the daisy chain arrangement where one slave is connected to the next, and the data flows serially from one slave to the next.

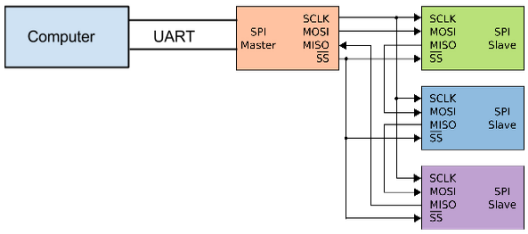


Fig 4.1 - Master Slave Arrangement

The KL25Z is severely constrained by the amount of memory in SRAM it has. Given that each LED required 3 bytes of information to light up properly, driving a large number of LEDs can result in a large memory footprint. The data for the entire image cannot, hence, be loaded into the master microcontroller in one cycle. To overcome this problem, the data for the image should be split into separate parts - where each part is a portion of the image to be driven by each microcontroller. This is done by the software on the computer. The software also tags each part with the address of the slave that has to drive that part of the image. Thus a packet consisting of the start byte, an address and the data is created. The packet is then sent from the computer over UART to the master. The master then sends the data through SPI to the slave. After receiving the packet, each slave examines the address part of the packet, and determines if the payload is for itself. If it isn't, it passes the same packet on to the next slave. This packet structure is shown below in figure 4.2.

Fig 4.2 - Communication Protocol

SPI inherently does not require an acknowledgement signal between the communicating parties. However, this can be the problem if the transmitter is sending out data, and the receiver is not listening. In order to overcome this problem, we needed to set up our own handshaking protocol over SPI. The protocol we have developed is shown above in figure 4.3.

The master sends the packet it receives from the computer to the slave. It then waits for the slave to send back an acknowledgement. If it does not receive this information in a certain amount of time, it will retransmit the packet. Once it receives acknowledgement from the slave, it will indicate the same to the computer, which will send the next packet of data. This process goes on till all the slaves have received their information, and then the master receives data corresponding to the LEDs it has to drive. Once it does, it will send a message to the slaves to start DMA transfer (which corresponds to start of driving of the LEDs).

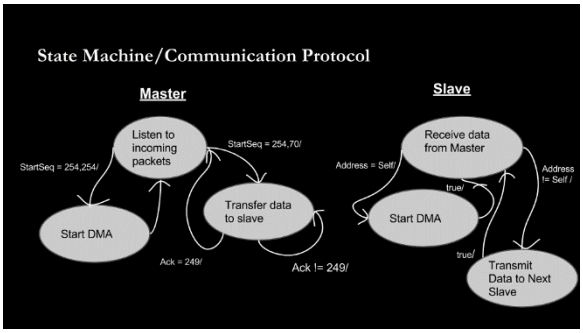


Fig 4.3 Communication Protocol State Machine

5. Simulator

Another common problem in embedded systems projects is that the test cycle is slow. This is because in order to be tested, firmware (FW) changes need to be deployed on crucial hardware. In our case, this was cumbersome, and many times not practical due to the following reasons:

1. Limited testable hardware: We had limited 16x16 LED modules, and only 1 30A power supply
2. Hardware Setup Cost: Wiring up the LEDs takes time, and is error prone

3. Hardware-related errors: Many times, the code seemed to not work, but in reality, the problem was a loose connection to the LEDs

So in order to speed up our test cycle, and follow good model-based design practice, we decided to build a LED matrix simulator that could give the developer feedback without the constraints of faulty hardware. The basic idea was to build a system that could monitor the lines going from the LED driver to the LED matrix to visualize the WS2812B protocol as an array of pixels on the computer. The basic process flow without the simulator is shown in figure 5.1 below:

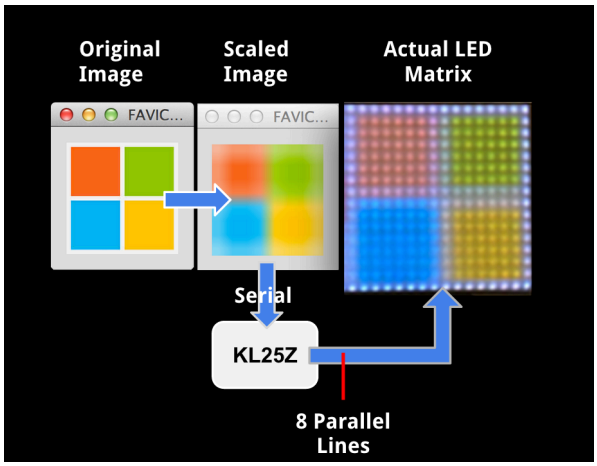


Figure 5.1 – Process flow for LED Matrices

With the simulator, instead of the lines going to the LED matrix, they will go to an external device where they are monitored by a Saleae logic analyzer. Figure 5.2 below shows this edited process flow.

The logic analyzer gives us the waveform corresponding to the image to be displayed on the actual LED matrix hardware in WS2812B protocol format. The last step in the process is to export this waveform data into CSV, and visualize this protocol using Processing. This workflow can be seen below in figure 5.3.

For our initial studies, we worked with the MS logo. The simulator does not work quite perfectly yet, but seems to be structurally similar to the MS logo it represents. With a little more time, this simulator should work well enough to replace hardware during tests. We spent a sizeable amount of time investigating using the KL25Z as a logic analyzer. With help of this open source FW and this open source logic sniffer we successfully turned a single KL25Z into an 8-probe, 2 MHz sample rate, 14 kB sample logic analyzer. Unfortunately, since the WS2812B protocol runs strictly at 800 kHz and has low pulse times at greater than 2 MHz, this approach did not work. It was, however, an interesting avenue to explore.

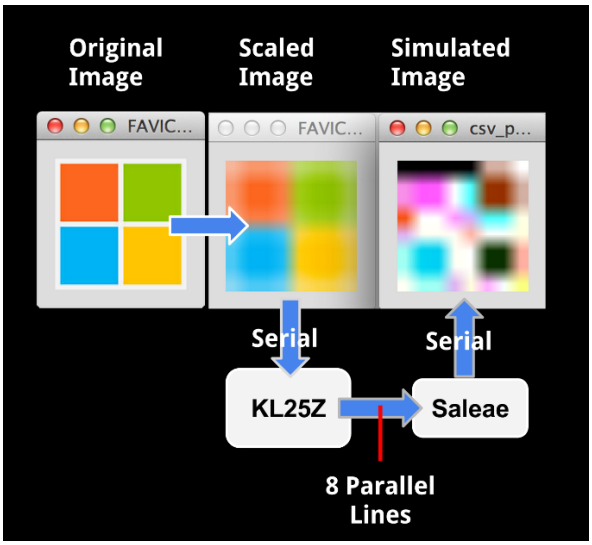


Figure 5.2 – New process flow using Saleae logic analyzer.

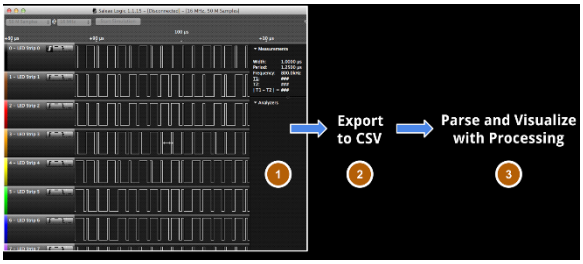


Figure 5.3 – Saleae protocol analyzer workflow.

6. Conclusion

Through this project, we laid the groundwork for the development a robust, scalable, and modular design to control LED matrices via a simple user interface. Our prototype demonstrates this by enabling the user to quickly setup and draw any image they choose on a plethora of setups, while the software tool keeps track of their setup to ensure that the user does not damage hardware. In

effect, we show that it is possible to apply good design principles to a low volume project, and to use these principles to create an abstract framework that can be applied in various other scenarios.

Future work continuing this project most notably include the development of a working simulator that can directly tie into the feasibility analyzer. This would allow for not only user validation of an image displayed on their workspace, but also for the rapid determination of whether or not displaying an image is feasible given the current setup.