NOV | 2013

VMS Display For ITS

Neethling McGrath

Design and build a prototype of a variable message system to display traffic notifications on the overhead message signs on a highway. The system must make use of the SMART platform to distribute the messages to potentially multiple VMS displays. A means to confirm delivery and successful display of the message is required. MTN and Trintel sponsor the project.

Report submitted in partial fulfillment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at the University of Stellenbosch.

Neethling McGrath
Stellenbosch University Engineering Faculty E&E Dept
Study Leader: MJ Booysen

Acknowledgements

Special thanks to:

Thinus Booysen - Guidance during the project

Ashley Cupido - PCB manufacturing

Edwe Cowley – Assistance on Modem and AT commands

Buckley McGrath – Guidance during project

Andreas Pfeffer - Manufacturing of LED display

Cornel Verster – Assistance on Trintel Smart Platform

Ivan Vorster – Report

Declaration of Own Work

I, the undersigned, hereby declare that the work contained in this report is my own original work unless indicated otherwise.

Name……………………………Signature…………………………..Date……………

Summary

Variable Message Sign (VMS) is an electronic display driven by various lighting solutions that include light bulbs and more recently ultra bright light emitting diodes (LEDs). Intelligent Transport Systems (ITS) refers to the application of advance technologies in the fields of electronics, communication, control systems and sensing to improve safety and efficiency in traffic situations through transmission of real time information. VMS are used as visual aid to relay real time information and functions as a component of the larger system.

The focus of this project is to design a VMS display that couples with a larger pre-constructed ITS, communicating through a GSM modem. The GSM modem connects to the SMART platform setup by Trintel to simulate the ITS and provides a graphical interface to input information to be displayed.

Opsomming

Variable Message Sign (VMS) is n elektroniese skerm wat gebruik maak van gloeilampe en, meer onlangs, ultra helder glimdiodes (Ultra Bright LEDs). Intelligente vervoerstelsels (ITS) verwys na tegnologie in die velde van elektronika, kommunikasie, beheerstelsels en sensors wat gebruik word om vervoer veiliger en meer doeltreffend te stuur deur die oordrag van inligting. Die VMS word gebruik as ‘n visuele hulpmiddel om informasie grafies te bied en vorm deel van die groter stelsel.

Hierdie projek fokus op die ontwerp van die VMS wat koppel met die reeds opgestelde ITS stelsel deur te kommunikeer via n GSM modem. Die GSM modem koppel die VMS aan Trintel se Smart platform wat opgestel word om die ITS te simuleer. Dit gee ‘n grafiese koppelvlak om inligting te kan opdateer.

Table of Contents

List of Abbreviations

LED – Light Emitting Diode
M2M – Machine to Machine
ITS – Intelligent Transportation Systems
VMS – Variable Message Sign
GSM – Global System for mobile Communication
SANRAL – South African National Road Agency Limited
CMS – Changeable Message Sign
PWM – Pulse Width Modulation
AT – Attention
SPI – Serial Peripheral Interface Bus
ASCII – The American Standard Code for Information interchange
DC – Direct Current
USB – Universal Serial Bus
MSB – Most Significant Bit
GPS – Global Positioning System

List of Figures

List of Tables

Chapter 1: Introduction

I. Introduction

Variable message sign (VMS) displays offer the ability to communicate real time information to ensure safer and more effective road environment. This is achieved by relaying real time information concerning the road and traffic conditions to drivers. Recently, the use of VMS throughout South Africa has become popular due to the push by the governments mandate to reduce road accidents and to increase operational efficiency of national roads.

Intelligent Transport Systems refer to the application of electronic and communication technologies to improve traffic flow and safety. The ITS monitors road conditions and communicates relevant information to road users or traffic authorities. The VMS links into the ITS that is operated by SANRAL (or similar authorities). It serves as a method to relay information to road users. This ensures the safe and effective flow of traffic. The communications of the ITS with the VMS is simulated by the Trintel Smart platform that constructs a Dashboard with various Gadgets that are used to input data that is to be displayed.

The scope of this project is to design and build a VMS prototype. The device communicates via a GSM modem with Trintel's Smart online platform. The platform relays only two lines of text that will be displayed on the device. The device will also be responsible for relaying operating states and acknowledge that it has received the data.

The display consists of a matrix of ultra-bright LEDs controlled by an arrangement of constant current sink drivers and constant voltage drivers. The drivers are controlled by an Arduino Mega microcontroller.

Data is communicated to the device through a serial connection to a GSM modem. The GSM modem will download the two lines of text from the Smart platform as data is sent by the ITS. The two lines of text are set on a Dashboard gadget and saved into variables on a webserver to be gathered by the modem as stated above. The microcontroller will periodically send status messages and also acknowledge once a message has been received so that the ITS can monitor the current messages on display.

The report firstly introduces the reader to previous work done on the field and literature that has been revised. Thereafter chapter two discusses possible concepts that could be used to achieve the project requirements. Chapter three discusses in detail the design choices that were made and how the device was constructed. The results of the project are discussed in chapter four. Finally chapter five advises recommendations and concludes the report.

II. Literature study

ITS and VMS:

The scope of the project does not include the operation of the ITS, however a communication medium from the intelligent transport systems to the VMS will be done through the use of the Trintel Smart platform. The ITS is responsible for the monitoring and reporting of traffic incidents, weather, congestion and various other events that could have an effect on the safety and efficient flow of traffic. The ITS utilises the VMS to report information that has been gathered through monitoring. The VMS is thus only a method of displaying information in a way that is understandable and functional to road users.

The VMS displays real time information relayed from the ITS to provide information to road users. There are various types of VMS technologies including permanent, portable and vehicle mounted. There are also Changeable Message Signs (CMS). The scope of the project focuses on permanently mounted and portable VMS's that are mounted on the side of the road or on overhead gantries or on vehicles such as trailers [1].

VMS capable of a variety of messages, depending on how the ITS has set up its procedure. The VMS will report to road users real time traffic information in a way that that is concise, unambiguous and prevents confusion. Accident messages will be used to warn traffic of incidents on the road and suggest alternatives. Congestion messages will display delay time and advise alternative routes. Roadwork and environmental conditions such as weather will only warn road users [2].

The messages will need to be in a general format that is both understandable while still using abbreviations and short sentences. The basic structure to follow is a problem, location, effect, action and then attention statements. The problem statement will alert of an accident. The location is self-explanatory however we require shorthand to present this in an easily readable and understandable format. The effect statement will indicate the result of the problem and how it affects the road user. The action statement informs the user what to do with the information presented by the VMS. The attention statement identifies the users who this problem is directed to. Thus, an example of a VMS message reads as follow: Congestion; Ahead; Minor delays; Find alternative Route; Buses [2].

VMS also have the option to not only use text messages but also relay information through pictures. This is seen as a way to transfer information quicker and also avoid translation issues completely.

Road users experiences regular problems with VMS use on roads. The main problem with the VMS is that road users only have a very short period of time to read the message, understand it and take action. This has to happen without confusing the user or drawing his/her attention away from maintaining control of the vehicle. To resolve this issue a few regulations have been put in to place. Messages that scroll or flash must be avoided as this will distract and confuse the users. The use of abbreviations is encouraged, however this must be standardised [3][4]. The length of a message must also be controlled as this could cause users to focus on the display for too long or not understand the message that was presented. VMS must also be positioned in a manner that is regulated too avoid confusion and increase visibility [5].

The messages sent to the VMS device has to be credited by the ITS. This is outside of the scope of this project. However the device must be able to display the messages in the format that is regulated by the ITS and must be included in the design of the project [4].

SANRAL requirements:

a) Policy:

SANRAL's traffic management centers operate the VMS in South Africa. They have developed a specific policy in regards to the content that may be displayed [17]. This policy requires that messages be easily understood. Messages need to be reviewed and acknowledged by the traffic management center. There has to be a message hierarchy that ranks the importance of certain messages above others. All messages will be displayed in English, as it is not practical to facilitate all the languages of road users especially in South Africa. A set structure has to be followed for messages as this promotes consistency and confidence in the VMS by the users. VMS will be left blank if there is no content to be displayed, however one continuously flashing LED will be displayed to ensure road users the device is functional. A standard set of messages will be used unless a special situation occurs where the supervisor will follow a set protocol. SANRAL will subcontract the operation of VMS and this policy requires that any operator adhere to these rules.

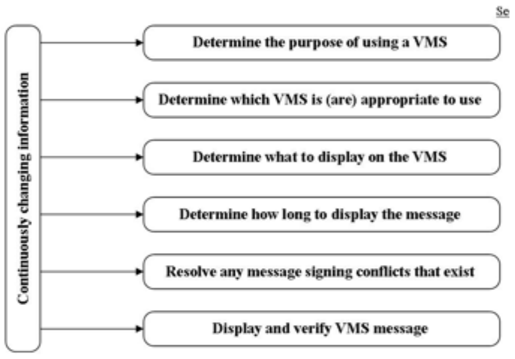

Sc
Continuously changing information
Determine the purpose of using a VMS
Determine which VMS is (are) appropriate to use
Determine what to display on the VMS
Determine how long to display the message
Resolve any message signing conflicts that exist
Display and verify VMS message

Figure 1: SANRAL VMS Procedure [17]

b) Display:

The actual time it takes a driver to read and process a message on a VMS is a function of the driver's speed, text formatting and the distance of the sign to the driver. The SANRAL policy investigates the driver's performance and found the following. These test are based on freeway speeds and reading distances of 44-94 meters [17].

Table 1: Read and process time of drivers at different speeds [17]

Speed in Km/h	No. Of Words	Time
80	8	2.94
100	7	2.54
120	6	2.14

c) Content Restrictions:

No event-sponsored names will be used and special event messages must be traffic related. No advertising, political or personal messages are allowed. No phone numbers greater than five digits, no websites and no SMS addresses are allowed. Even though other countries encourage the use of graphical messages SANRAL has decided against this and it is not allowed. Abbreviations need to be used sparingly as this could cause confusion and lower the trust of road users in the message content. Only recommended abbreviations are allowed [17].

Lighting Source- Ultra Bright LED's:

VMS technology was traditionally designed to utilize light bulbs as a lighting source, however this presented a few problems. Light bulbs do not last long if they are toggled and also require a large amount of power. The industry started looking at alternatives and identified LEDs as a possible solution. This was not a feasible option at the time as LEDs were not able to provide enough luminance to replace light bulbs to power large format displays. Over time, LED technology has improved and was able to create devices with high luminance, higher durability and lower power requirements than light bulbs. These are essentially the only requirements for a VMS application.

The LED is a type of diode that uses the basic concept of semi conduction. Semi conduction refers to the ability to conduct current in one state and in another totally isolate two conductors. This essential characteristic is achieved through the use of materials with specific characteristics. A pure conduction material is mixed with the impurities of another material. This process is called doping. Pure material atoms bond perfectly, leaving no free electrons to carry current, creating an isolator. An impure material (created through doping) creates free electrons that are capable of carrying the current and holes or the absence of electrons that allow electrons to flow through the material. The semi conductor with extra electrons is called a N-type material. The semi conductor with extra holes is known as a P-type material. Bonding a p-type material next to an n-type material creates a diode (see figure 2).

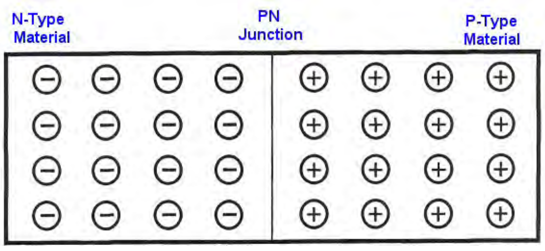


Figure 2:PN Junction

Applying a positive voltage on the n-type material pulls the electrons to the terminal creating a depletion zone on the P-N junction, thus no current flows. Applying a positive voltage on the P-type junction injects electrons into the holes and current flows, provided the electrons are extracted at the n-type material. Thus, through this setup of semi conducting materials a device is created that only conducts current in one direction (see figure 3).

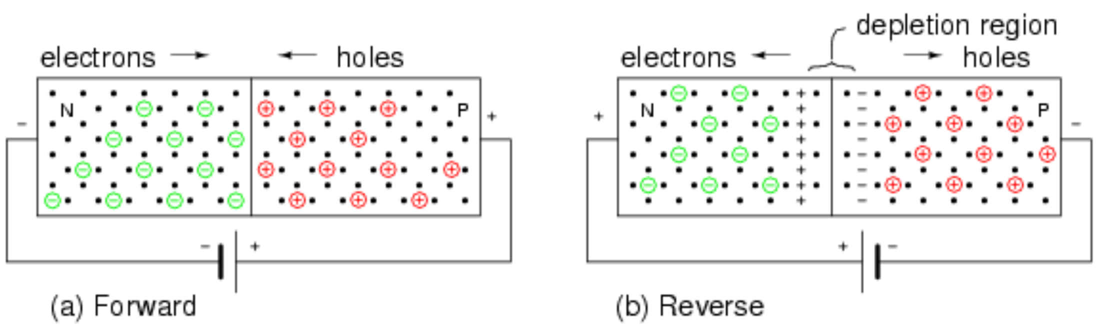


Figure 3: Electron Flow in Junction

The Voltage-Current curve characteristic of a diode is determined by the flow of electrons through the P-N junction. If a large enough forward voltage is applied to reduce the size of the depletion region the flow of current is instant. However, if the forward voltage is not large enough to reduce the size of the depletion region no current will flow. If a large enough reverse bias voltage is applied the diode experiences breakdown where a large current flows in the opposite direction. This is due to the creation of a large amount of hole and electrons. This will damage the device permanently.

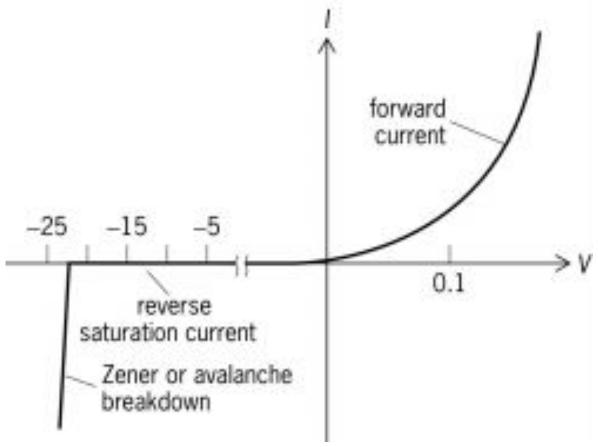


Figure 4: Diode V-I characteristics

A LED contains a crystal made from Gallium, Arsenic and Phosphorus that contains a large number of atoms and electrons. The electrons are arranged in different energy levels around the atom and the furthest electrons can easily escape the pull of the nucleus. As the electron is rotating around the nucleus it carries a certain amount of energy. The electron can gain additional energy through heat or electricity. This requires the electron to jump to a higher energy band (see figure 5). At this stage the electron is unstable and releases the energy through a light photon.

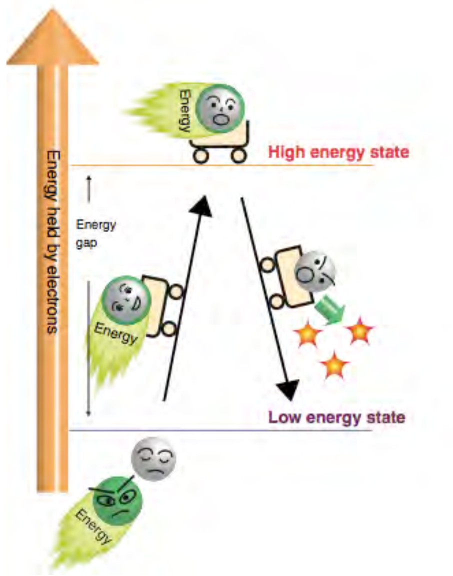


Figure 5: How a photon is released [7]

Similar to diodes, the n-type semiconductor has a large number of free electrons. However, in the case of the light emitting diode, the electrons are in an unstable position. The N-type semiconductor consists of the Gallium, Arsenic and Phosphorus crystal. The P-type semiconductor consists of an excess of holes created by the Zinc atom. As the two materials are joined to form a P-N junction holes and electrons diffuse into the opposite material. Once a source is connected to the light emitting diode, electrons move from the n-type and join with holes in the p-type and release the excess energy in the form of light (see figure 6). Different color LEDs are constructed through using a different crystal. By changing the materials that are used to construct the crystal the wavelength of the photon released is changed, thus changing the color [7].

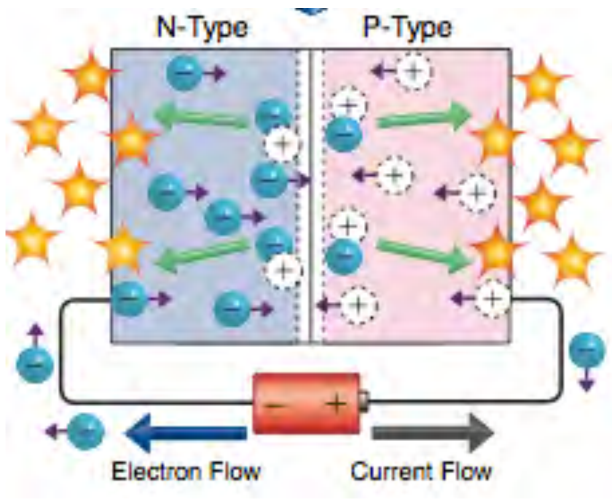


Figure 6: LED Junction [7]

The brightness of a LED is directly related to the amount of current flowing through it. Thus the larger the current, the brighter a LED will illuminate [8].

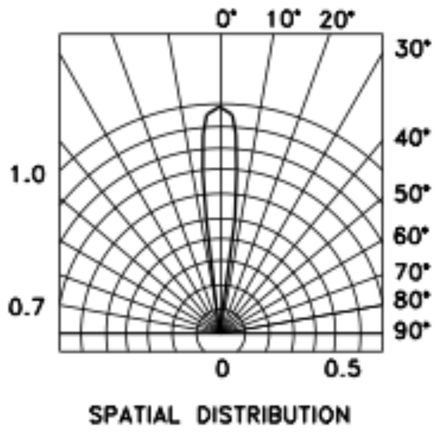


Figure 7: LED photon distribution [8]

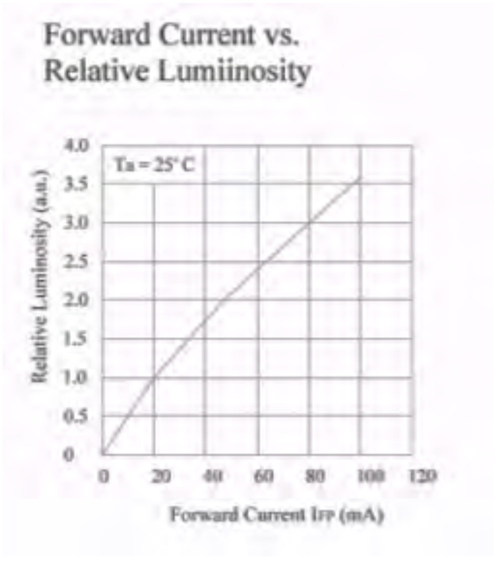


Figure 8: LED Light intensity relative to current [8]

LEDs appear brighter when pulsed at higher current levels because the human eye acts as both an integrator and peak detector. Utilising this unique capability of the eye, pulsing the LED at a very low duty cycle but high intensity (high current) makes the LED appear active for longer [8].

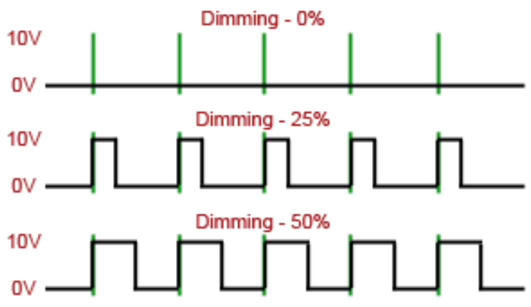


Figure 9: PWM Signal

Pulse Width Modulation (PWM) allows for the LED to be driven at a higher current. This is due to the fact that the average current over the entire cycle will still be below the maximum allowed. This means that a LED can be pulsed on at a higher luminance for a shorter period [8][9].

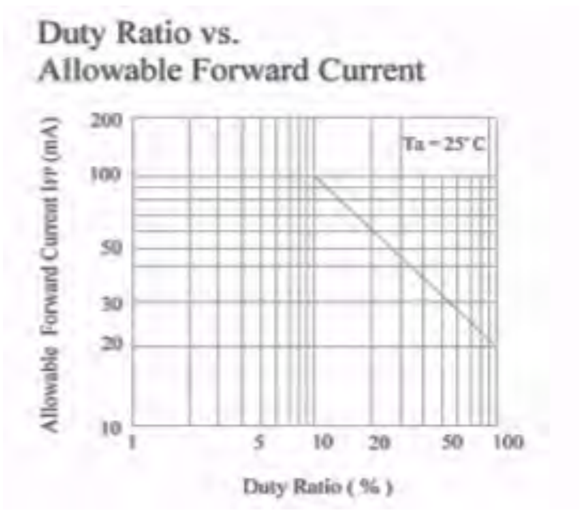


Figure 10: LED Allowable current Duty Cycle [7]

Matrix display:

In order to display messages LEDs are placed in a dot matrix arrangement. All characters can be formulated by a specific combination of an 8x8 matrix of LEDs through this arrangement. Every LED therefore needs to be individually controlled and be independent of the state of any other LED in the matrix. Therefore each LED must be addressed individually. There are various schemes by which to solve this problem. They will be discussed under the concept design chapter.

Trintel Smart Platform:

Trinity's Smart platform powered by Sierra Wireless is a Machine-to-Machine (M2M) solution to remotely monitor devices through the use of a GSM modem. M2M is defined as the communication of two electronic systems without any human intervention. The goal is to enable two devices to share data between each other autonomously [12]. In this project Trintel's Smart Platform will be used to collect data and deliver commands through the use of a cellphone network [12]. Sierra Wireless's Airvantage portal offers a method of gathering data from devices through a modem connected to the platform. Through the use of the Airvantage Configuration tool a user is able to customize the portal to specific requirements. It has the ability to create variables, setup alerts and manipulate data. The Airvantage platform updates data to the SMART platform. The Smart platform provides the option of manipulating data and presenting it in a graphical interface for users to edit variables [11].

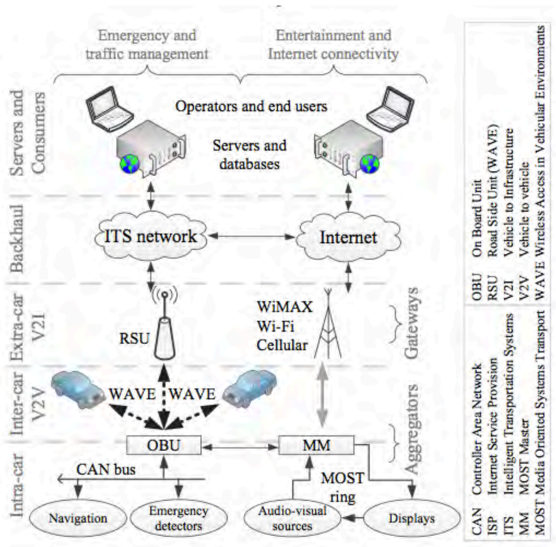


Figure 11: Basic M2M Network [10]

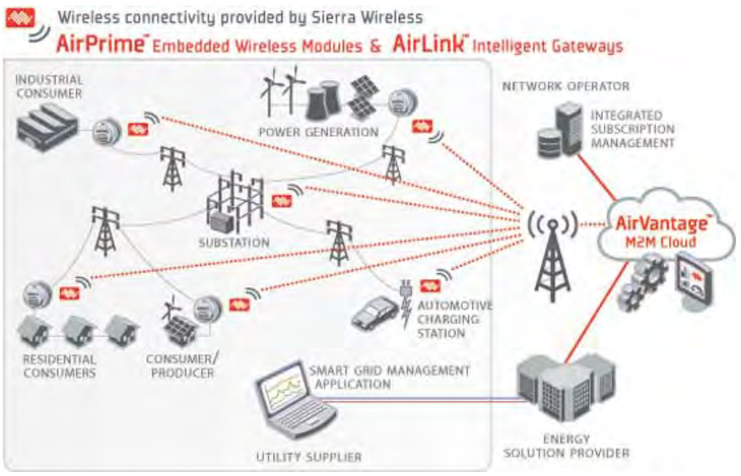

Wireless connectivity provided by Sierra Wireless
AirPrime Embedded Wireless Modules & AirLink Intelligent Gateways
INDUSTRIAL CONSUMER
POWER GENERATION
NETWORK OPERATOR
INTEGRATED SUBSCRIPTION MANAGEMENT
SUBSTATION
AirVantage M2M Cloud
AUTOMOTIVE CHARGING STATION
RESIDENTIAL CONSUMERS
CONSUMER/ PRODUCER
SMART GRID MANAGEMENT APPLICATION
UTILITY SUPPLIER
ENERGY SOLUTION PROVIDER

Figure 12: Sierra Wireless M2M Network [11]

The Airvantage Configuration tool is used to setup the asset that is used to define the variables, command, event and alarms. This file is then uploaded to the Airvantage platform and associated with a specific device. There can be more than one asset associated with one device [11].

The SMART platform links low level variables to metric models. The metric models can then be used to manipulate variable data and present it in a graphical form through the use of gadgets.

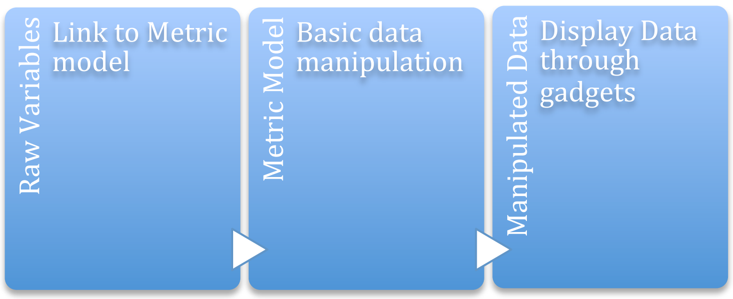


Figure 13: Sierra Wireless design process of Asset [11]

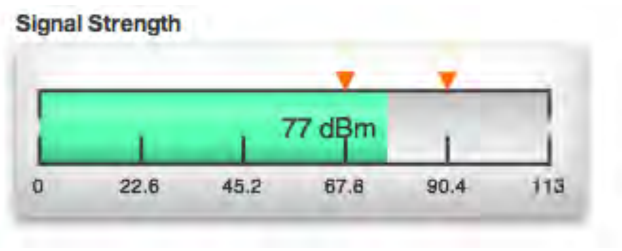


Figure 14: Trintel Gadget [11]

FXT 009 Modem:

The Sierra Wireless modem communicates through serial communication to the device. Custom versions of AT commands are used to call specific operations. The modem is used to transmit data between the Airvantage servers and the device through a GSM connection [11].

Figure 15: Sierra Wireless FXT 009 model [11]

Chapter 2: Concept Design

I. Hardware

Adressing:

The main problem presented by a large number of pixels (LEDs) is finding a method to address each pixel individually; this is referred to as direct addressing. It is costly and impractical to have a data line for each pixel. This requires a controller to have as many ports as pixels without expanders. However, it does have the benefit of being able to constantly keep each LED illuminated, so there is no flickering [14].

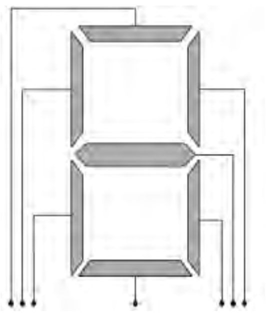


Figure 16: Direct addressing [15]

There are two methods of solving this problem:

a) Passive Matrix:

Passive matrix arrangement multiplexes the LEDs in a matrix with a common row line and common column. Passive Matrix multiplexing is cost effective and the most popular addressing method used in VMS. The disadvantages include crosstalk due to the row pixels being electrically related and minimal multiplexing capability. Addressing is achieved by selecting row-by-row scanning through the entire display. Once a row, is selected current is imposed in all pixels in the row. By selecting each pixel in a column the device is able to display information. A voltage source is used to supply voltage on the anode of the LED and a current sinking element is connected on the cathode [15][16].

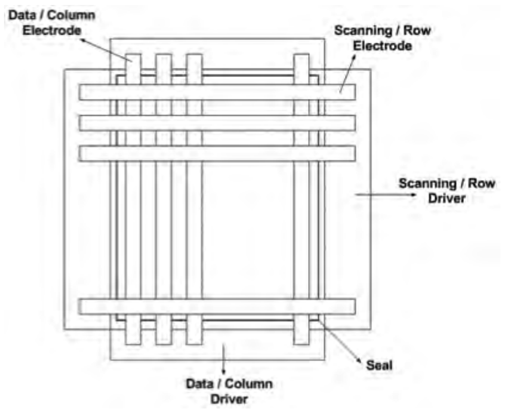


Figure 17: Passive Matrix Addressing

b) Active Matrix:

The Active matrix arrangement uses the exact same multiplexing approach as the passive matrix. However, there is a switch element on each pixel that provides a 100% duty cycle. The element is switched on by pulses received from the row and column that keeps the LED on during the entire cycle. This solution completely removes all crosstalk and enhances contrast. However, it is more expensive, as it requires a switching element on each pixel [15][17].

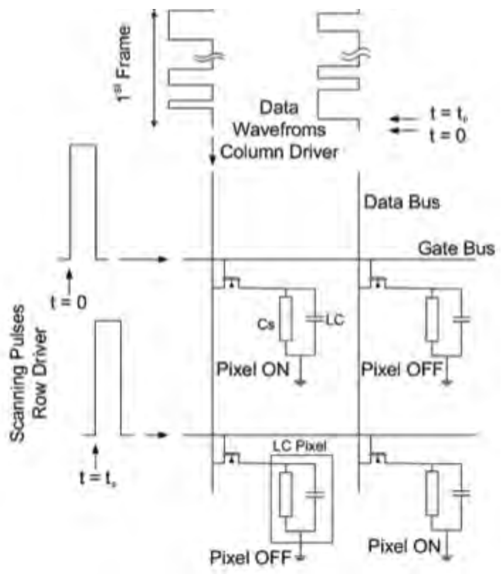


Figure 18: Active Addressing [15]

Through the use of the multiplexing approach the data lines are significantly reduced.

$$\begin{gathered} Direct\ Adressing\ Datalines\ =\ no.\ of\ pixels \\ Passive/Active\ Matrix\ Datalines\ =\ Rows\ +\ Columns \\ no.\ of\ pixels\ =\ Rows \times Columns \end{gathered} \tag{1}$$

LED Powering Circuits:

LEDs are constructed out of semiconductor materials that together form a P-N junction. These materials form a voltage potential difference across the junction. When the LED is forward biased a current is able to flow through the element and it emits light. The voltage required to bias the element is called the biasing voltage. The voltage ranges from 1.5 V to 3 V due to different materials used for different color LEDs. Thus in order to switch a LED the element requires a forward voltage that biases the device and a current path that provides enough current as the luminosity of the LED is directly proportional to the current flowing through it [8].

$$I_f = \frac{V_{CC} - V_f - V_{Switch}}{R}$$

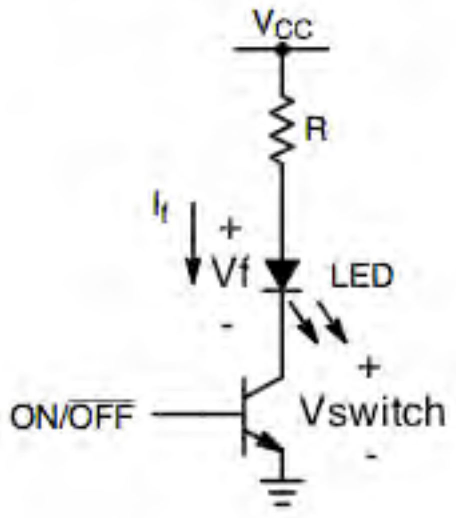


Figure 19: Basic LED Circuit [8]

LEDs Voltage-Current characteristics require that current needs to be limited by another element in the circuit. If a large amount of current is flowing through the LED junction for too long the element will overheat and be permanently destroyed. The simple solution is to place a resistor; this will cause a voltage drop and control the current in the chain. As discussed above, the problem of addressing requires that there be a similar circuit for each LED. This becomes very complex as the number of LEDs increases. Integrated circuits, (called drivers) are coupled to the column and rows of the displaying. As there are two methods of addressing the types of drivers used for each will be discussed.

a)   Passive Matrix Driver:

Passive matrix arrangements requires that all the LED's anodes in a row be coupled, the cathodes will be coupled in the columns.

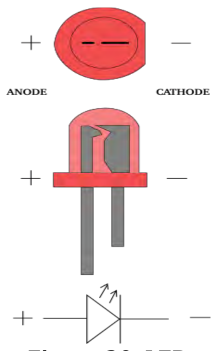


Figure 20: LED

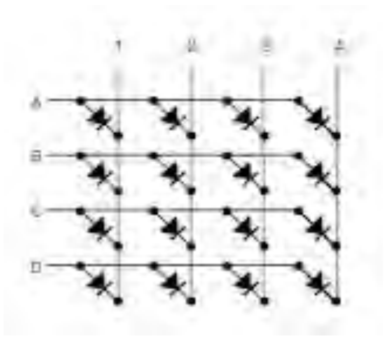


Figure 21: LED Matrix [18]

By placing a forward biasing voltage on each row in a sequence the circuit only needs the cathode to be connected to a current sinking element. A Voltage Source Driver on the anode (Rows) and a Current Sinking Driver (Column) is required for this setup.

The Voltage Source Driver needs to be able to supply sufficient current for the row. The max current will occur when all LEDs in the row are switched on.

$$Voltage\ Source\ Driver\ Current\ req\ =\ LED\ Pulse\ Current\ *\ Number\ of\ LEDs\ per\ row \qquad (2)$$

The Voltage Source Driver must also be able to provide a large enough voltage potential on the anode to bias the LED so that it switches on. An amplifier is the element that is used to achieve these requirements focusing on current gain. The Darlington Pair setup is ideal as the current gain by the first amplifier is amplified further by the second stage. As this element will be responsible for sourcing the current the NPN arrangement is fitting. The second stage emitter node will be connected as output to our row. A Darlington pair is required for each row to supply current and voltage.

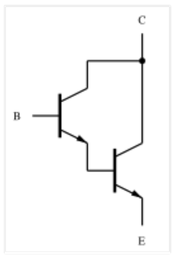


Figure 22: Darlington Pair

The current sinking driver will serve the purpose of completing the path for the current to go to ground. The current sinking drivers will be attached to the cathode of each LED in the column. This requires an independent current source that is able to absorb a constant current no matter what load (voltage) is present on the output terminal. Thus, it must be independent of voltage and able to sink a constant current.

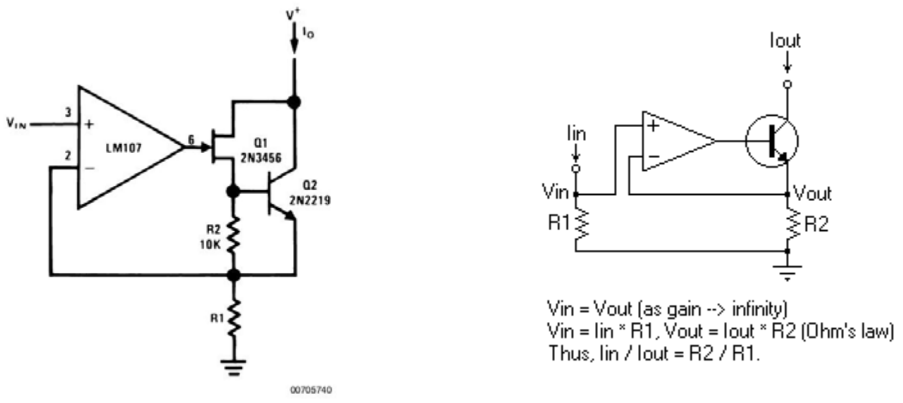

V+
Io
VIN
LM107
Q1
2N3456
Q2
2N2219
R2
10K
R1
00705740
Iout
Iin
Vin
R1
Vout
R2
Vin = Vout (as gain --> infinity)
Vin = Iin * R1, Vout = Iout * R2 (Ohm's law)
Thus, Iin / Iout = R2 / R1.

Figure 23: Current Sinking Circuits

It is important that the current sink maintains a constant set current as (in this setup) this function will restrict over current through the LED and protect it from overheating. Once again there will be one constant current sink driver per column. Figure 23 shows two possible solutions to achieve these requirements. The input reference current is mirrored on the output. The output current is scaled due to the input voltage, due to reference current and R1. Thus the output current is a scaled version of the input reference current.

b) Active Matrix Driver:

The active matrix driver focuses on the newer technology of Organic Light Emitting Devices (OLED) displays. These displays are thinner, lighter, no viewing angle restriction and more efficient than current LED solutions. As discussed the active matrix arrangement requires a switching element for every pixel [18]. These switching elements (usually a Thin film Transistor) are addressed through the columns and rows to activate individual LEDs. Thus the drivers are only responsible for toggling the switching device on every LED; the switching device controls the current through the LED.

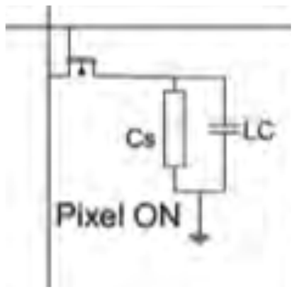


Figure 24: Active Addressing Pixel [15]

Through active addressing the LED is able to stay active for a longer due to the thin film transistor having a capacitor attached to keep it conducting for the entire frame. Once the LED is toggled it remains switched on until it gets addressed again. A 100% duty cycle is achieved. This is a significant advantage over the passive matrix addressing. However this is more expensive and not needed for this projects application as the advantages are focused on visual quality and contrast, none of which is particularly important to the VMS application.

c) Shift Registers:

Matrix arrangement is a method of reducing the number of data lines. LED powering drivers do not all need to be active at once yet must be independent of one another. The voltage source drivers will need to scan through the matrix row – by – row. The current sinking drivers will be used to select each individual LED in the row that needs to be active. Coupling shift registers to activate the driver outputs reduces the number of data lines and allows the use of SPI communication. Shift registers are basic integrated circuits that consist of flip-flops in cascade. A flip-flop is a basic element capable of storing a voltage potential. The flip-flops share a common clock where the data of the next flip-flop is connected to the previous flip-flop. Once the clock triggers the data shifts through the cascade of Flip Flops, this results in the data shifting one flip-flop down the sequence (see figure 25). By connecting the outputs of a shift register to the voltage source driver gating signal, the entire driver can be controlled by only one data line and one clock line. This is also true for the constant current sink driver.

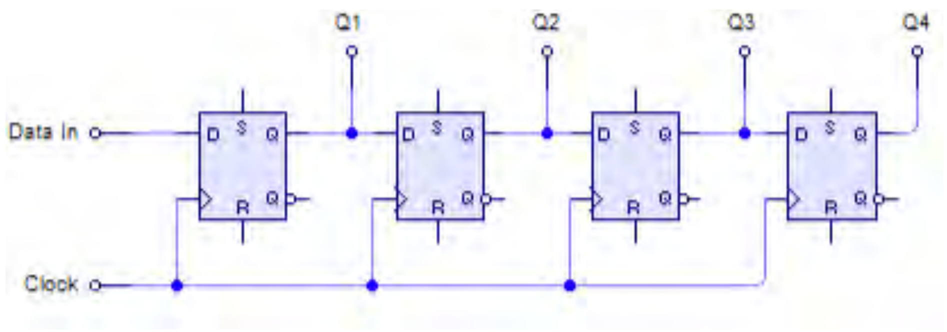


Figure 25: Arrangement of Flip Flops in Cascade

Communications with ITS:

Additionally this project requires that a message is sent back to the ITS acknowledging that a message has been received and is currently being displayed. This will ensure that the ITS is aware of what is being displayed on the VMS at all times.

II. Software:

Serial Periphiral Interface - SPI:

The flip-flops will control the drivers. To toggle the outputs to a specific sequence the data needs to be clocked into the flip-flops, Serial Peripheral Interface (SPI) will be used. SPI is a synchronous communication protocol that requires four data lines; input and output data, clock and slave select. Data can be sent to the shift registers through the data lines and expanded to the drivers through individual flip-flop outputs. The SPI channel needs to be configured depending on the speed of transmission, polarity of the clock and the most significant bit of the data.

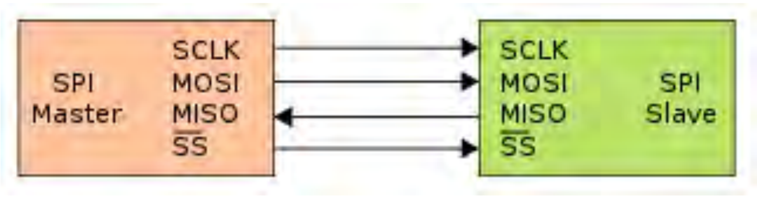


Figure 26: SPI Wiring

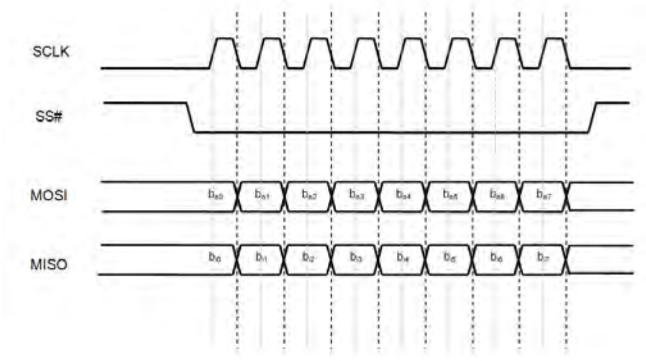


Figure 27: SPI Logic Levels

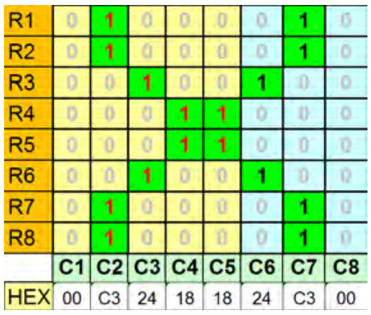
Controlling the display:

The hardware is now capable of addressing every LED individually. The voltage and current drivers are controlled via the SPI channels. The data sent to the drivers now needs to be generated by a controller. As discussed the voltage source drivers will scan (row-by-row) through the entire display sequentially as fast as possible to reduce visibility of flickering on the screen. Only a single row will have a voltage bias at a time. Once the row has been selected the constant current drivers will ground the anodes of LEDs addressed to be on. On the controller a two-dimensional Boolean array will represent the matrix display.

The controller will thus constantly be required to transmit data through the SPI channel to keep the display up to date.

The controller must be able to clock data fast enough over the SPI channel such that no flickering can be seen . It will also require two serial data ports and two SPI ports to support the peripherals of the project. The memory specifications require that the controller be able to store the two dimensional Boolean array as well as vital state machine variables.

The Modem will deliver the string variable from the platform in the form of an ASCII message. This will be saved in the controller but will need to be formatted, as it needs to be presented on the display matrix. A conversion from ASCII to Dot matrix is required. Firstly the individual dot matrix size per character needs to be determined. There are various sizes, each with advantages in quality and drawbacks in the number of LEDS required. It was decided that an 8x8 Dot matrix would be used per character. A lookup table will be used to point the controller to the Dot Matrix representation of a specific character. It will require some time to format the entire display array in the controller and a delay can be expected between receiving data and the data being displayed on the VMS.



R1	0	1	0	0	0	0	1	0
R2	0	1	0	0	0	0	1	0
R3	0	0	1	0	0	1	0	0
R4	0	0	0	1	1	0	0	0
R5	0	0	0	1	1	0	0	0
R6	0	0	1	0	0	1	0	0
R7	0	1	0	0	0	0	1	0
R8	0	1	0	0	0	0	1	0
	C1	C2	C3	C4	C5	C6	C7	C8
HEX	00	C3	24	18	18	24	C3	00

Figure 28: Dot-Matrix representation of X in HEX

M2M and the Trintel Smart platform:

M2M application is used in the form of a Sierra Wireless modem attached to the VMS device. This modem is constantly and autonomously communicating with the Smart Platform through a GSM network. The Smart Platform will be required to manage the data required by the device. This includes the string variable used to store the message that will be displayed and a system status Boolean. The Platform must also graphically allow an operator to edit the string variable and display a confirmation message. The AirVantage Configuration tool is used to setup the Asset as is required by these specifications [10][19].

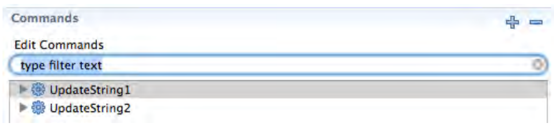


Figure 29: List of Commands on Trintel Platform [12]

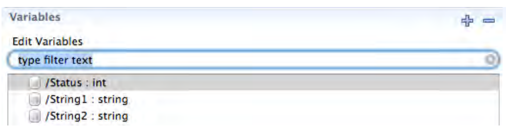


Figure 30: List of Variables on Trintel Platform [12]

Chapter 3: Design

For this particular application of LED displays the most important features are visibility and reliability. The VMS needs to be visible from far away and in various weather conditions and operators must be confident in the system's ability to function at all times. Both addressing systems have features that contribute to visibility however the active addressing is aimed more at consumer level television screen rather than VMS application. The passive addressing scheme was chosen due to its ability to achieve clear visibility and remain cost effective. The hardware will be designed to be able to display two lines of 10 characters. An 8x8 LED matrix will represent each character adding up to 80 columns and 16 rows. Each frame will thus consist of 1280 LEDs. The controller must be able to scan through all 16 rows fast enough so that no flickering is noticed; a frame rate of at least 30 fps is required. The controller must also be able to present data quick enough to the column drivers such that the row drivers scan timing is not affected (see figure 71).

I. Hardware:

LED:

The Kingbright Super bright 10 mm LED was chosen as lighting source. This LED junction consists of Gallium Aluminum Arsenide and produces a red light. A 2.5 V forward biasing voltage activates the LED. The LED is capable of a maximum forward current of 20mA DC and 155mA peak current.

Figure 31: LED

Controller-Arduino Mega:

The controller used is the Arduino Mega. This is open source hardware developed in 2005 in Italy by student of Interaction Design Institute Ivrea. The Mega 2560 is the specific Arduino used and is a microcontroller based on the ATmega2650. It has four UART ports for serial communication. The project requires two. It operates at a clock frequency of 16 MHz and has enough inputs and outputs to control the row and column drivers. The controller is powered by a DC adapter or through USB [20].

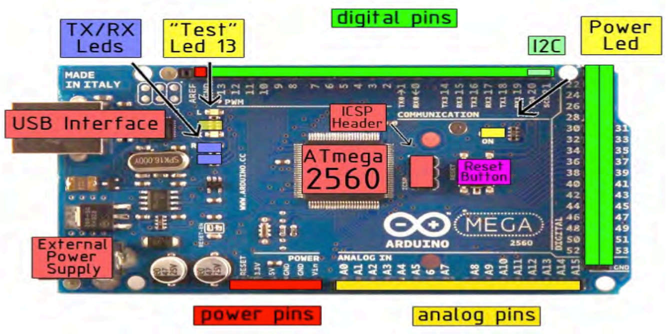


Figure 32: Arduino MEGA [20]

Unfortunately the Mega only has one hardware implementation of SPI. Thus, another needs to be programmed in software. This could be problematic as the speed of the software implementation is significantly slower than the hardware implementation. The Mega also contains 256 KB flash memory that will be more than required in this implementation. The Mega is programmed via the Arduino Environment that uses a hybrid of C-code known as Wiring. Wiring is an open source electronics platform that uses an IDE also known as Wiring to support the C coding language [20].

LED Matrix:

The proposed design consists of 14 column and 10 rows. Only the LED display is constrained by these restrictions. The drivers and software supports the full display size. The prototype display is constructed by drilling 140 10mm holes into a square piece of plastic. The LEDs are then placed into the holes. Each row's anodes are connected together and each column's cathodes are also connected.

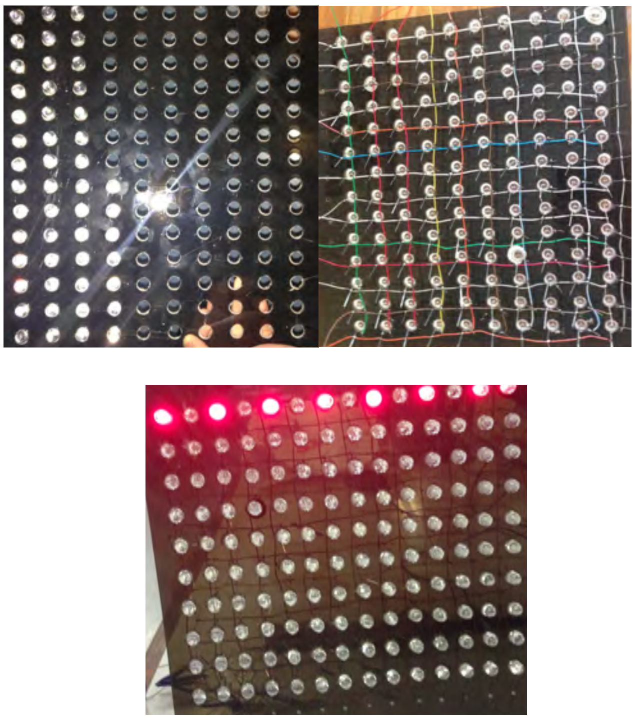


Figure 33: LED Display Matrix Construction

Voltage Source Driver:

The Voltage Source Drivers need to provide at least a 2.5 V potential on the anode of the LED being powered. Additionally, it must be able to source 20 mA per LED. Even though the LED is capable of sustaining more current. Testing the LED at various current levels it was found that the LED becomes blindingly bright at higher current levels. Current can be increased if required. The VMS is designed for 80 LEDs in each row, thus the driver must be able to supply enough current in the case that all are switched on.

$$Source\ Current\ =\ 20mA\ *\ 80\ =\ 1.6\ A \tag{3}$$

Thus the voltage source must be able to provide a maximum of 1.6 A.

The voltage source driver that was chosen is the MIC5891 made by Micrel. The MIC5891 is a high voltage, high current latched driver. Each individual driver output consists of eight Darlington pair transistors. The Darlington pairs are controlled by CMOS circuitry; this includes the SPI, strobe and output enable pins. The driver functions at 5 V logic supply voltage and can receive data at a maximum clock frequency of 5 MHZ. The driver is capable of sourcing up to 35 Volts on the output pins of each driver that is more than sufficient for this specific application. 8-bit Parallel shift registers control the drivers and data is clocked in via SPI. Each chip is capable of delivering 500mA per channel if only one channel is active at a time. This is ideal, as the hardware requires only one row active at a time. However, the current supply required for the display is 1.6 A. Three MIC5891 integrated circuits are required to power 8 rows (there will always be one LED per character per row that is off due to the text formatting see table 4) resulting in a effective number of 70 LEDs per row and 1.4 A [21].

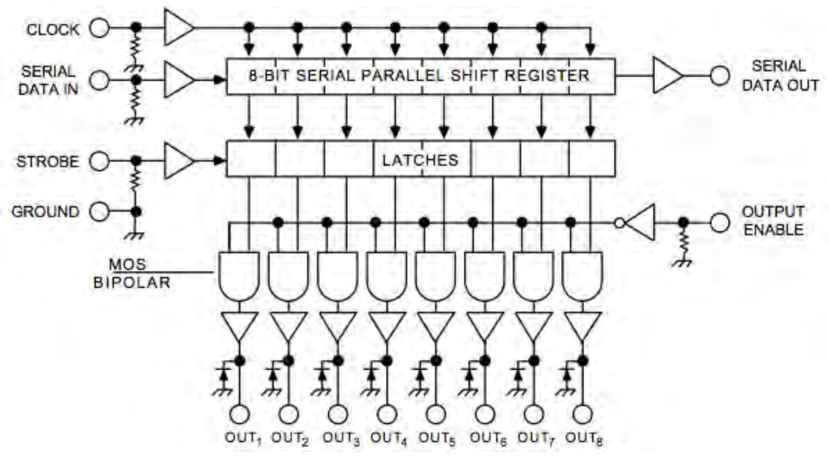
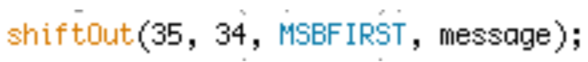


Figure 34: Voltage Source Driver [21]

The VMS display size is constrained to 16 rows, thus requiring two sets of three MIC5891 parallel integrated circuits. A total of MIC 5891 drivers are used. The two sets of three MIC5891 are connected in cascade with the first chips Serial Data Out connected to the next Serial Data Input. The data is transmitted to the 8-bit shift registers on the rising edge of the clock. When the strobe pin is toggled from high to low the data in the shift registers are transferred to the output buffers [21].

The MIC 5891 driver uses software implementation of SPI. A clocking frequency of 60 KHz was measured. This emulation is restricted to only clocking in data on the rising edge, the data was sent most significant bit first (MSB). The voltage source driver only has one channel active at a time. In the figure one can see that channel 10 is currently being addressed. Once all 16 bits of data has been clocked in the latch line is toggled.



```
shiftOut(35, 34, MSBFIRST, message);
```

Figure 35: Shiftout Code

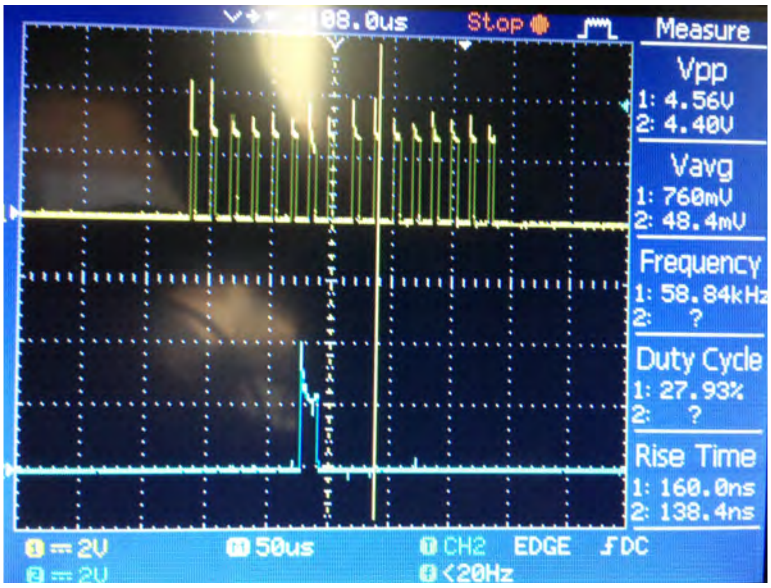


Figure 36: Voltage Source SPI Data Line

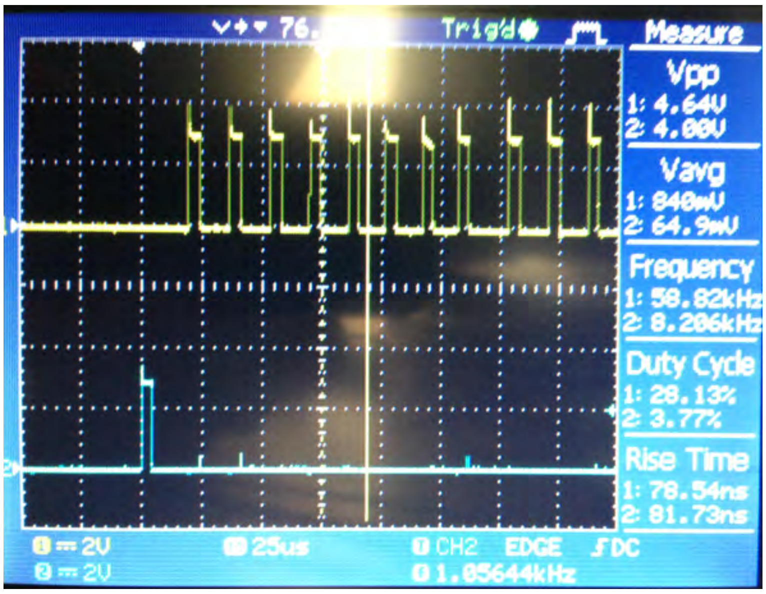


Figure 37: Voltage Source Latch line

Constant Current Driver:

The constant current driver is responsible to complete the circuit and ground the cathode of the LED while limiting the current on the pre-discussed 20mA. The project has 80 LEDs in a column that needs to be controlled. The driver chosen is the Texas Instrument TLC 5926. The TLC5926 has 16 constant current driver outputs. An external resistor sets the constant current. The IC requires a 5 V voltage supply and data is clocked in at a frequency of 30 MHZ. A 16-bit parallel shift register and latches are used to convert the serial data that is received into parallel format that is used to control the output drivers. The output buffer is capable of supporting up too 17 V on the output buffers. The high speed clocking frequency also meets the high data volume data requirement. There are 16 drivers per TLC5926 and the display requires 80 drivers, thus resulting 5 chips being used. The external current limiting resistor is calculated as 1k Ohm [22].

$$I_{OUT,target} = (1.25\ V/R_{ext}) \times 15 \qquad (4)$$

$$R_{ext} = 1\ K\ Ohm$$

$$I_{OUT,target} = 18.75\ mA$$

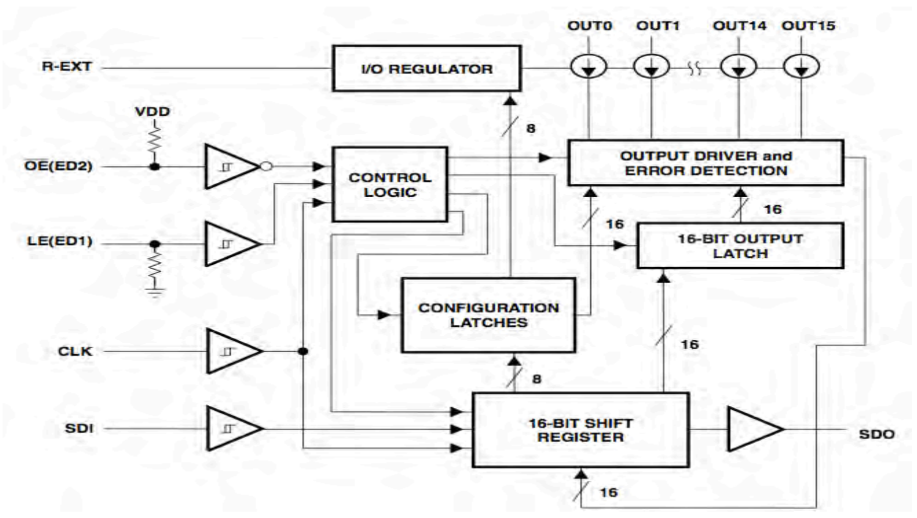
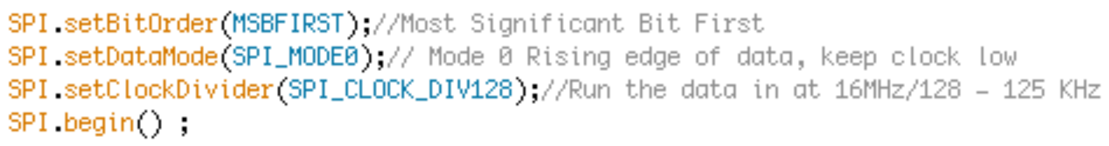


Figure 38: Contract Current Driver [22]

Data is transmitted to the Serial Data Input (SDI) pin from the controller using SPI into the 16-bit shift registers using the rising edge of the clock. The Latch (LE) pin is used to transfer data from the shift registers when the pin is toggled high to low. The output enable (OE) pin controls the output drivers. When OE is low the output drivers are active. The SPI bus uses a 5 V logic level. The five TLC5926 chips are connected in cascade with the first Serial Data Output connected to the Serial Data Input of the next. This results in 80-bits of data required to control the entire rows LEDs [22].



```
SPI.setBitOrder(MSBFIRST);//Most Significant Bit First
SPI.setDataMode(SPI_MODE0);// Mode 0 Rising edge of data, keep clock low
SPI.setClockDivider(SPI_CLOCK_DIV128);//Run the data in at 16MHz/128 - 125 KHz
SPI.begin() ;
```

Figure 39: SPI setup Code

The program is written with the convention of clocking in the MSB first on the rising edge of the clock. The clocking frequency is calculated by dividing the Arduino clocking frequency by 128, this results in a clock frequency of 125 KHz. All 80 bits are sent sequentially and one can clearly see that only the lowest 16 channels are used in the prototype. The data line is held high during no transmission. The latch line is only toggled once all 80 bits of data has been shifted through.

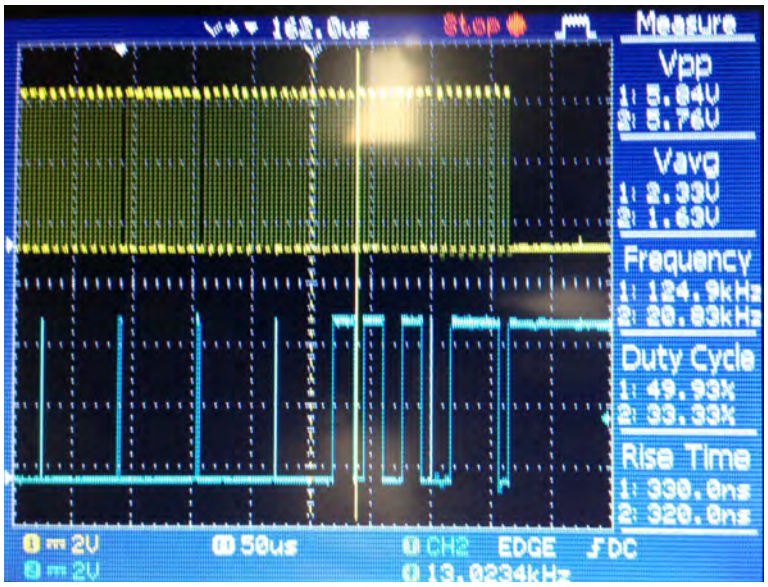


Figure 40: Constant Current Driver SPI Data Line

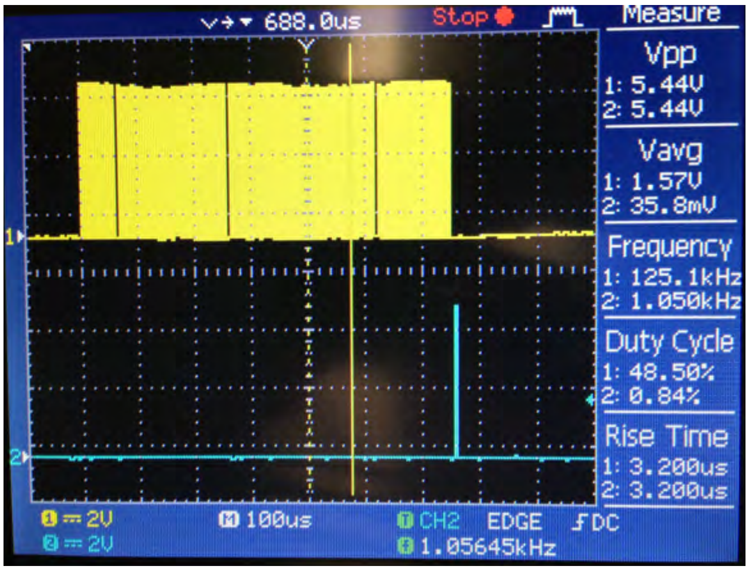


Figure 41: Constant Current Latch

Modem:

As discussed, the project requires the use of the Sierra Wireless FXT 009 GSM modem sponsored by Trintel. The modem uses an RS-232 connector to communicate with the controller. The modem uses only ASCII AT commands to transmit variable and control data. A wall outlet powers the modem. The modem connects to the Trintel platform through an edge connection supplied by a gateway from MTN [13].

The controller expects 5 V logic levels on the serial ports for communication, however the RS232 connection functions on 12 V. In order to convert the voltages for the controller and vice versa, the Digilent Pmod232 was used. The Pmod232 uses the Max3322 chip to achieve the voltage conversion [23].

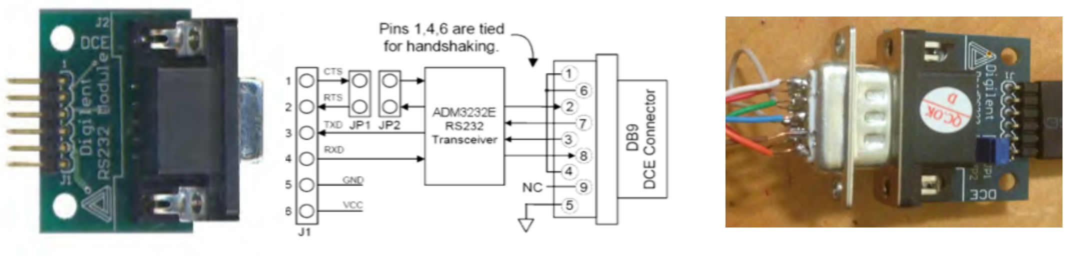


Figure 42: Pmod Wiring and Contruction [23]

AT Commands:

AT commands are used to control the modem. The commands are versatile and can report back system status, queue variable updates, allow commands to be reported and acknowledge message received. There are a number of commands that can be sent [24]. The modem is setup to communicate serially at 9600-baud rate.

This table shows a summary of the AT commands sent by the controller:

Table 2: AT commands sent by Controller

AT command	Function
AT+IPR=<rate>	Set the baud rate of the serial communication of the modem
AT+AWTDA=d, "<Asset Name> ", Number of Variables, "Variable name, Type, Value"	Updates a variables on the Smart platform with specific value
AT+AWTDA=c, "<Asset Name> ","<Command Name>"	Notifies the Trintel platform that a command is ready to be received
AT+AWTDA=a, "<Asset Name> ", <Id>	Transmit an acknowledge message to notify the Trintel platform that a command has been received

This table shows a summary of the AT received from the modem:

Table 3: AT commands received by controller

OK	AT command sent is correctly received and in the correct format
ERROR	AT command not in correct format
+AWTDA: BOOT	Modem is currently booting up
+AWTDA: UP	Modem currently running and connected
+AWTDA: DOWN	Modem currently running but is not connected

II. Software:

Trintel Smart Platform:

The Smart platform requires that all basic level variables first be converted into metric that the platform understands and can manipulate. There are various transforms that can be applied to the variables before linking them to the metric. Transforms include basic formulas, Timer difference calculators, Base station identifiers, GPS location, user input, data input/output and finally commands. The commands transform is the only one of worth to the project as it is used to trigger the download of the string variable to the device. [12]

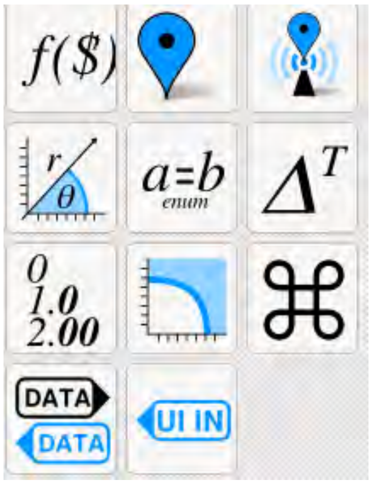


Figure 43: Trintel Transforms [12]

The UpdateString command had to be split into two because the system does not allow connecting two user inputs to the same command. This one line of the VMS is connected to one command. The second line is connected to the second command. The Metric was programmed to connect the UpdateString command and user input transform to the Command transform and finally couple it to the Metric UpdateString1. The same process was followed for UpdateString2.

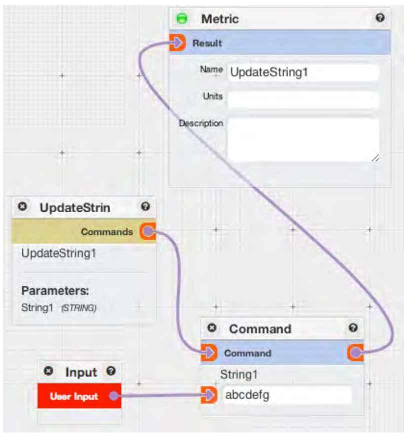


Figure 44: Trintel Metric Construction [12]

The Metrics are then coupled with the text input gadget that allows the user to manipulate the string variables. The Dashboard (see figure 45) is then setup to present the operator with the two text box gadgets that represent the two lines of the VMS. The user edits the text boxes and saves the data. Once the device communicated with the modem via AT commands that it is ready to accept the data, the platform transmits the two lines of strings together with a unique job id tag. This tag is received by the device and relayed back in the form of a acknowledge message to confirm the command was received and is currently displaying the message that is on the platform. The system state gadget indicates the current state of the system. State one indicates a green status. All connections are active and display running. State two indicates a yellow status. This means that communications have been lost but is still displaying content. State three indicates a red status. The system is rebooting no content is being displayed at this time.

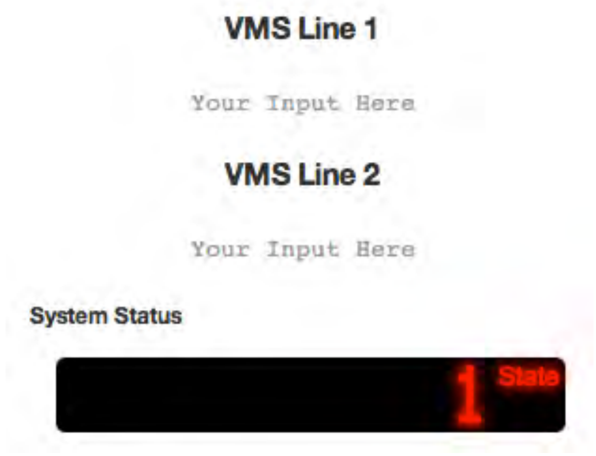
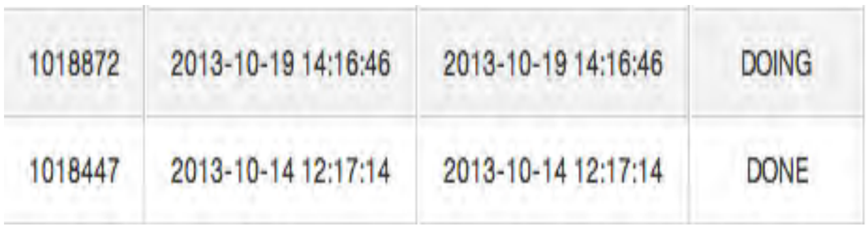


Figure 45: Trintel Dashboard [12]

The operator can validate that an acknowledge message was receive by checking the jobs tab (see figure 46) and waiting for the status to be "DONE". "DOING" indicates that the command has been sent but no acknowledgement message has been received yet. The first column data is the unique ID that has to be sent back by the device.



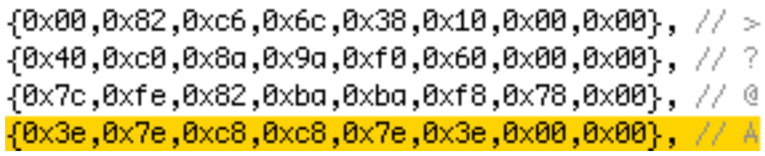
1018872	2013-10-19 14:16:46	2013-10-19 14:16:46	DOING
1018447	2013-10-14 12:17:14	2013-10-14 12:17:14	DONE

Figure 46: Trintel Job List [12]

Text Formatting:

An 8x8 Dot matrix char array lookup table was generated to represent the on and off states of each LED depending on what character it is supposed to graphically construct.

The lookup table is a constant character array with 128x8 dimensions. The code uses the row index as a direct conversion from char ASCII data to an integer. This method thus only requires that the controller convert the ASCII value directly to an integer to reference the eight lines of 8-bit Dot matrix data [reference ASCII table].



```
{0x00,0x82,0xc6,0x6c,0x38,0x10,0x00,0x00}, // >
{0x40,0xc0,0x8a,0x9a,0xf0,0x60,0x00,0x00}, // ?
{0x7c,0xfe,0x82,0xba,0xba,0xf8,0x78,0x00}, // @
{0x3e,0x7e,0xc8,0xc8,0x7e,0x3e,0x00,0x00}, // A
```

Figure 47: Lookup Table

Here "A" represents the 65th row in the array. The data in each column of the 65th row represent each column of the Dot matrix representation of an "A". The result of the data graphically can be seen in Table 4.

Table 4: Visual representation of Lookup table data of an "A" on its side

0	0	1	1	1	1	1	0	0x3e
0	1	1	1	1	1	1	0	0x7e
1	0	0	0	1	1	0	0	0xc8
1	0	0	0	1	1	0	0	0xc8
0	1	1	1	1	1	1	0	0x7e
0	0	1	1	1	1	1	0	0x3e
0	0	0	0	0	0	0	0	0x00
0	0	0	0	0	0	0	0	0x00

The Trintel Platform sends the command with the two strings and the controller formats each character into dot matrix format and places it into the display matrix.

Program Flow:

The controller runs a state machine with four states.

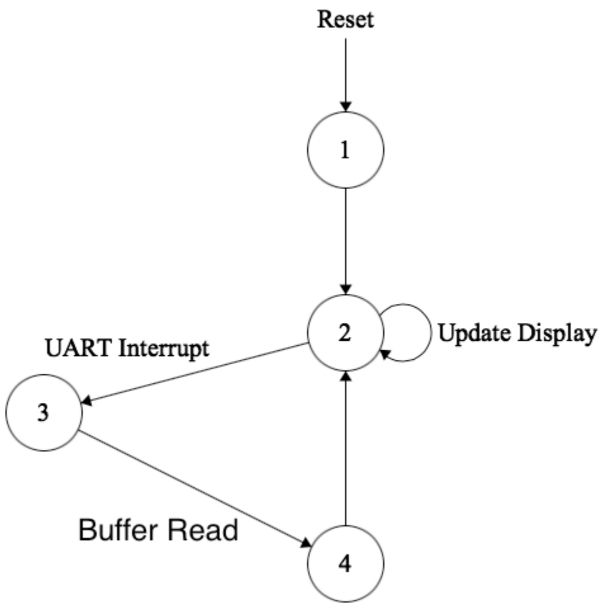


Figure 48: Finite State Machine

a) State 1 Initialize:

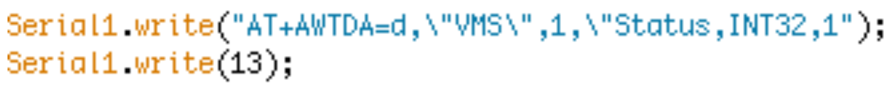
The first state is responsible for the initialization of the entire device as this is the first state that is entered after the hardware resets. The serial communication of the modem and the debugging console is set up according the predetermined baud rates. The SPI bus is set up to send the most significant bit first (MSB) on the rising edge of the clock that is set to 4 MHz. All pins that are required to communicate with the different driver chips are also set up as output pins and initial states set. All registers are cleared and variables initialized. The controller's timer is set up. Finally the modem is tasked to write that the device is ready to receive the two commands from the Trintel Platform. Once the initializations are done the program automatically jumps into the second state.

b) *State 2 Main Program Loop:*

The main loop constantly calls the updateDisplay() function. This function is the heart of the operation and must be the main routine. It must constantly be serviced for the display not to start flickering, interrupts subroutines must be kept short. The function steps through the rows and transmits the data that is in the display matrix array. The row driver's data requires two 8-bit bytes that are transmitted through a software emulation of SPI that is somewhat slower this is transmitted every time a row is selected. The column driver's data is transmitted 8-bits at a time through the SPI bus until all 80 bits in the selected row's columns constant current drivers shift registers has been filled. Once the data has been transmitted the controller latched the data to the output buffers and the LEDs in the row are activated. The program then steps to the next row and repeats the process until the entire frame has been transmitted. Once this is done the main program loop restarts and the display is updated from the first row again. The main loop can be interrupted by the UART due to incoming data from the modem signaling that the Trintel Platform has sent a command or status message. In this case the program moves into state 3.

c) *State 3 Communications Received:*

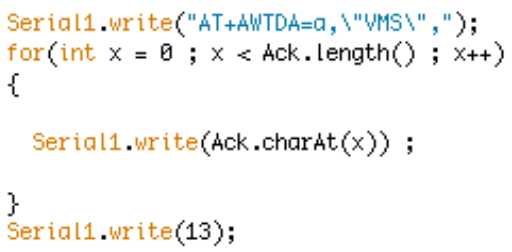
Once the UART triggers the interrupts, the program enters state 3 where the message in the buffer is interpreted. The function stringFormat() is used to decide whether the message is the command that is received or a modem status message. If the messages contain status messages the modem is tasked to update the state to the online platform with the following AT-command.



```
Serial1.write("AT+AWTDA=d,\"VMS\",1,\"Status,INT32,1");
Serial1.write(13);
```

Figure 49: Modem Write Command

This command updates the status variable on the platform to the value "1", indicating the system is running correctly. If however a command is received the data required is extracted from the message, this being the string data and the unique acknowledge id. The acknowledgement message is then tasked to the modem as follows.



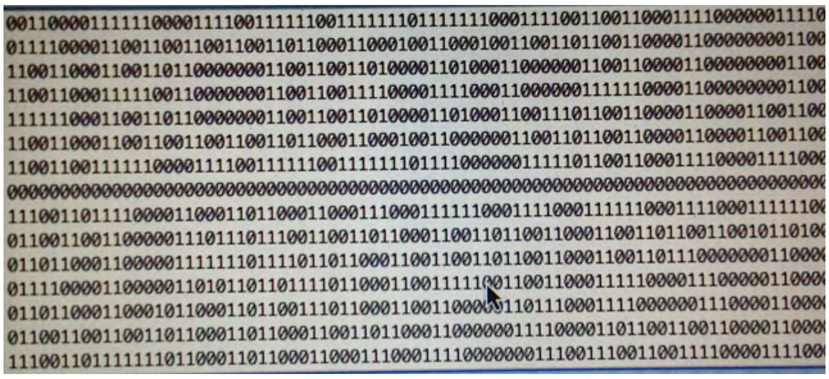
```
Serial1.write("AT+AWTDA=a,\"VMS\",");
for(int x = 0 ; x < Ack.length() ; x++)
{

  Serial1.write(Ack.charAt(x)) ;

}
Serial1.write(13);
```

Figure 50: Write Acknowledge Command

Once the acknowledgement message has been sent the program jumps to state 4 where the display matrix is updated.

d) State 4 Display Matrix Update:

The string data has been saved globally and requires to be formatted from character ASCII data into the dot matrix format of 8x8 per character. The function updateDisplayMatrix() is used to achieve this. The function takes each line of the VMS and steps through character-by-character using the lookup table to write the data into the specific slot in the display matrix. This is done each time the VMS string data is changed, as the program has to avoid long subroutines in the main program loop.



```
0011000011111100001111001111110011111101111111000111100110011000111100000011110
01111000011001100110011001101100011000100110001001100110110011000011000000001100
11001100011001101100000001100110011010000110100011000000110011000011000000001100
11001100011111001100000001100110011110000111100011000000111111000011000000001100
11111100011001101100000001100110011010000110100011001110110011000011000011001100
11001100011001100110011001101100011000100110000000110011011001100001100000110011
11001100111111100001111001111110011111110111100000001111101100110001111000001111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11100110111100000110001101100011000111000111111000111100011111100011110001111110
01100110011000000111011101110011001101100011001101100110001100110110011001011010
01101100011000000111111101111011011000110011001101100110001100110111000000001100
01111000011000000110101101101111011000110011111001100110001111110000011100000011
01101100011000101100011011001110110001100110000011011100011110000000111000001100
01100110011001101100011011000110011011000110000001111000011011001100110000011000
11100110111111110110001101100011000111000111100000001110011100110011110000011110
```

Figure 51: Display Matrix Content in software

Important Functions:

a) updateDisplayMatrix();

This function is responsible for the updating of the DisplayMatrix array once the new messages have been received from the ITS. These messages then get converted into Dot-Matrix format and placed into the array.

The function begins by initialising a line counter, column counter and character counter variable. These variables are used to control the nested loops. The first counter loops through all ten characters of the first line until all characters have been converted into Dot-Matrix format. Once a character is selected by the outer loop another loop begins counting through the row lines of the character. The loop counts through 8 row lines because the project selected an 8x8 character format. Once a row line has been selected another loop counts through the columns and reads the Dot-Matrix data from the lookup table. Through this nested loop sequence each characters Dot-Matrix data gets formatted into the DisplayMatrix array. This process is repeated for the second line as well (see figure 63).

b) serialData();

This function is only used to interpret the AT commands received from the modem. It is a case statement with various actions depending on what commands the modem transmits to the device.

There are various status update messages depending on whether the modem is operating successfully. Also this function calls the updateDisplayMatrix(); function and transmits the acknowledgement message back to the trintel platform (see figure 64).

c) UpdateDisplay();

This function is responsible for the drivers containing the correct display data and needs to be serviced every frame. This function is called in the main program loop and must not be interrupted.

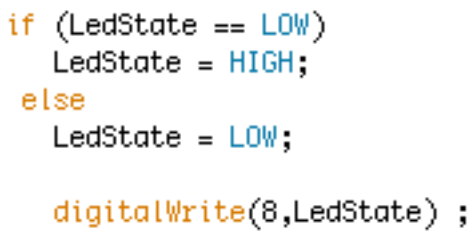
The outside loop counts through the rows and transmits 16-Bits through SPI to the voltage source drivers. Inside this loop is another loop that counts through all columns. 80-Bits of data is then sent through SPI to the constant current column drivers. Then the column and row driver's latch lines are toggled and data is transferred to the outputs. This process gets repeated for every row (see figure 61).

Chapter 4: Results

The VMS system performed as specified by the project requirements. The device is able to fully communicate with the ITS through the modem. The device transmits status data and acknowledge messages and receives the message data from the ITS. The display is capable of displaying all ASCII characters.

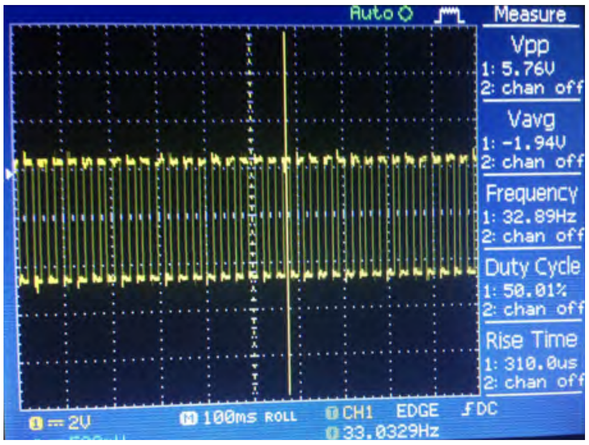
LED Display performance:

The LED display is capable of displaying the message content without any flickering and achieves a 66 Hz frame rate (shown in figure). The brightness of each LED is fully visible from a distance, however the viewing angle is very narrow. Unless the user is perpendicular to the display the brightness greatly reduces.



```
if (LedState == LOW)
   LedState = HIGH;
 else
   LedState = LOW;

   digitalWrite(8,LedState) ;
```

Figure 52: Main loop refresh rate test code



```
Auto        Measure
            Vpp
            1: 5.76V
            2: chan off
            Vavg
            1: -1.94V
            2: chan off
            Frequency
            1: 32.89Hz
            2: chan off
            Duty Cycle
            1: 50.01%
            2: chan off
            Rise Time
            1: 310.0us
            2: chan off
1 == 2V    M 100ms ROLL    T CH1  EDGE  DC
                           33.0329Hz
```

Figure 53: Refresh rate of display

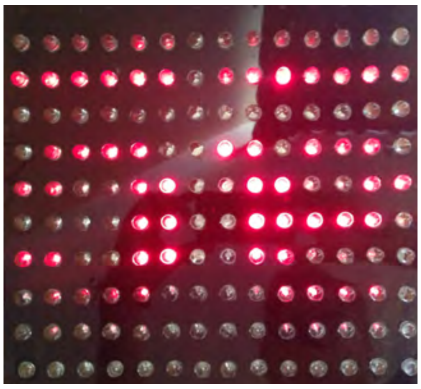


Figure 54: LED display perpendicular

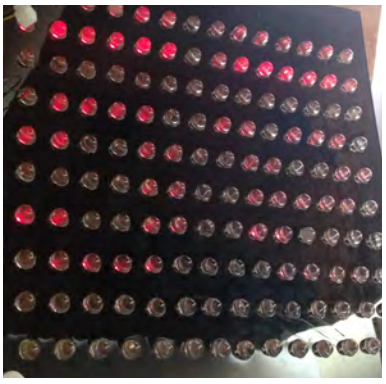


Figure 55: LED display off center

Due to the narrow viewing angle the use of dispersion film is required. This film is used to diffuse the light emitted from the LED. However this film greatly reduces the brightness experienced by the user.

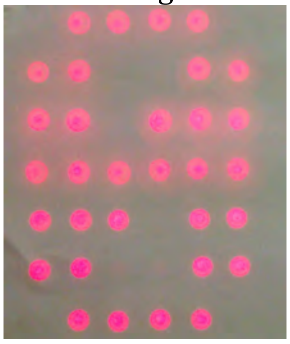


Figure 56: Display with simple dispersion film

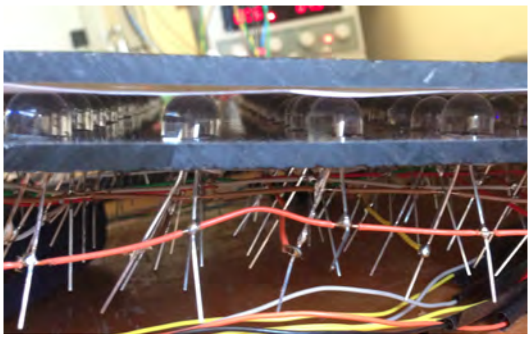


Figure 57: Display with Perspex dispersion

Modem commands received/sent:

The Trintel platform transmits and received a number of testing data to indicate the system is fully functioning. These are some of the message received by the device through the serial communications with the modem.

Commands:

The first line is the message received in AT commands from the modem followed by the message, the unique command id. The unique id is then acknowledged back to the platform to confirm that the message is received and is being displayed.

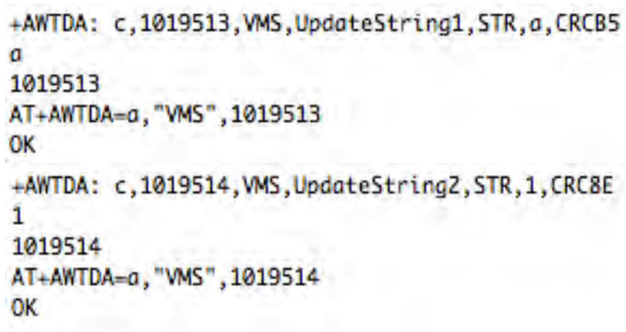

```
+AWTDA: c,1019513,VMS,UpdateString1,STR,a,CRCB5
a
1019513
AT+AWTDA=a,"VMS",1019513
OK
+AWTDA: c,1019514,VMS,UpdateString2,STR,1,CRC8E
1
1019514
AT+AWTDA=a,"VMS",1019514
OK
```

Figure 58: Serial communication Commands received

Status:

This indicates the device is operating in the green state, meaning there is communications and everything is running perfectly. This is then uploaded to the platform.

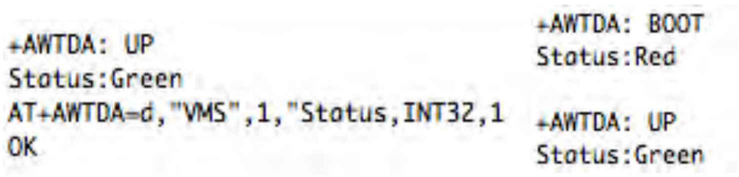
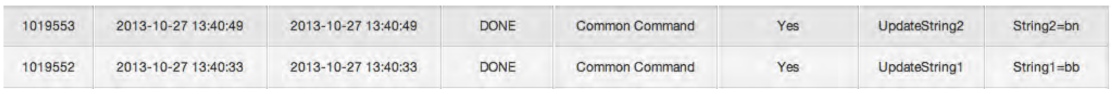

```
+AWTDA: UP
Status:Green
AT+AWTDA=d,"VMS",1,"Status,INT32,1
OK
```
```
+AWTDA: BOOT
Status:Red

+AWTDA: UP
Status:Green
```

Figure 59: Status message and AT command

Trintel Platform:

This indicates that the command has been sent from the platform to the device. The device received the command and acknowledged that it is displaying the message.



1019553	2013-10-27 13:40:49	2013-10-27 13:40:49	DONE	Common Command	Yes	UpdateString2	String2=bn
1019552	2013-10-27 13:40:33	2013-10-27 13:40:33	DONE	Common Command	Yes	UpdateString1	String1=bb

Figure 60: Smart Platform Jobs Tab

Chapter 5: Conclusion & Recommendations

I. Recommendations:

The Ultra-bright LEDs used in this project suffer from a very narrow viewing angle. This introduces an unforeseen problem in the need to use a dispersion film. The film however greatly reduces the brightness of the LEDs to viewers. If the use of a dispersion film is required the hardware allows for the LEDs to be driven at a higher current value however a current limiting resistor sets this. The As initially designed, the Voltage Source driver is only capable of delivering 20mA to each LED as it was initially designed. This can easily be solved by adding more MIC5891 chips in parallel, as there are already three in the current design.

There are however different options that can be pursued. The use of surface mount LEDs with much greater viewing angles is recommended. These LEDs do not offer the same brightness that the Ultra-bright LEDs offer. However considering this method does not require the use of dispersion film the resulting visibility of messages will greatly increase over longer distances. These LEDs can be used in place of the LEDs and will function in the matrix design used.

Due to the controller only having one hardware SPI bus either the column drivers or row drivers will need data transmitted by software emulation of SPI hardware. This is significantly slower than the hardware SPI. The controller therefore limits the frame rate not the drivers used. Even though one SPI bus would normally be sufficient for IC's with chip select lines, the drivers used do not posses this feature. Either a controller with two SPI hardware busses needs to be used or an improvised chip select line needs to be implemented to increase the frame rate past its current restriction of 66 Hz.

Implementation of graphical message content can also be added to the system. However, the online platform is limiting in regard to this. One possible solution is linking ASCII characters that are not used in messages to graphical message content saved on the device. However, this could possibly confuse the operator.

It is also recommended that more characters be added to each line, as this will increase the type of message that can be displayed. This can easily be done by daisy chaining more TLC 5926 chips and adding more MIC 5891 chips. The design is very modular with regard to the size of the display but there is a limit to the size caused by the speed of the SPI busses.

II. Conclusion:

The purpose of this document is to report detailed technical information on researched topic, concept proposal, design, construction and testing of the VMS project. The system transmits messages from an online platform to the device. The device then displays the message content, warning road users of possible hazards or various other message functions.

Chapter one gives an overview of previous work done on the topic. In this chapter the scope of the project was established and restrictions were identified. The chapter summarizes current methods of designing VMS and also regulations put in place to assist in the design process. In chapter two different design concepts were discussed. Different advantages and disadvantages are weighed and cursory explanations of the specific concepts are explored.

Chapter two introduces the requirements of this project and discusses possible concept designs. Chapter three discusses the chosen concept in-depth. The concepts design is explained in depth regarding the functionality, how it is connected and all components. This chapter completely explains why the concept was chosen in regards to its advantages that it possesses and why the disadvantages are acceptable. Chapter four presents the results of the project.

The VMS was able to successfully display all ASCII characters supported by the platform and is visible at a reasonable distance. However, this can be improved by using surface mount LEDs. All channels of both the voltage source drivers and the constant current sinking drivers are functioning properly. However, only the channels used by the prototype is drawing current. The display achieves 66 Hz frame-rate. This is sufficient to remove flickering completely. Once messages are sent from the platform it takes about 5 seconds to transmit the data to the device. Once the data is received the display instantly changes the data being displayed, there is no delay. The online platform operated by Trintel experiences regular timeouts and this results in messages not being queued, this is outside the scope of the project but could result in malfunctioning. The online platform dashboard displays the last messages sent and thus the message being displayed. Also the dashboard displays the current state of the device, operators are completely informed regarding the functionality of the VMS at all times. The device is completely powered off a 5 V voltage source. The prototype display required a maximum of 200mA depending on what message was being displayed.

This project achieved all the requirements set out in chapter two and three. Therefore, this project was successful in achieving the requirements. A video showing overview of the system and functionality can be found at http://www.youtube.com/watch?v=QlnRh9gu1D8.

References

[1] New York State Thruway Authority, "GUIDELINES FOR USE OF VARIABLE MESSAGE SIGNS (VMS)," Department of Maintenance and Operations, [May 2011]

[2] ALBERTO ARBAIZA, ANTONIO LUCAS-ALBA, "Variable Message Signs Harmonisation PRINCIPLES OF VMS DESIGN," EASYWAY, Trans European, [January 2012]

[3] S Clark, "Variable Message Signs," Government of South Australia, [June 2010]

[4] Kimley-Horn, "SANRAL Variable Message Sign (VMS) Usage Policy," The South African National Roads Agency Limited, [February 2010]

[5] Phil Margison and Michael Bushby, "Guidelines for the Location and Placement of Variable Message Signs," RTA, [December 2008]

[6] The South African National Roads Agency Limited, SANRAL Variable Message Sign (VMS) Usage Policy, SANRAL, 2010.

[7] Kazuo Murata, "How do LEDs emit light", OKI DATA Tech Tech Journal, [January 2004]

[8] KingBright, "10mm Solid State Lamp," L-813SRC-B [February 2008]

[9] Jim Lepkowski and Mike Hoogstra, "NL27WZ04 Dual Gate Inverter Oscillator Increases the Brightness of LEDs While Reducing Power Consumption," Arizona State University. AND8067/D [October 2007]

[10] M.J. Booysen et al, "Machine-to-Machine (M2M) Communications in Vehicular Networks," KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS VOL. 6, NO. 2, Feb 2012

[11] Sierra Wireless, AirVantage Platform, pp.1-4.

[12] Trintel, "Trinity-Smart Sight," [Online]. Available: http://smart.trintel.co.za [Accessed: Oct, 2013].

[13] Sierra Wireless, "Fastrack Xtend," AirLink FXT Series Report. WA_DEV_FEX20_UGD_00, 7 Apr. 2007.

[14] Walid Benzarti et al, “Compact Analytical Physical-Based Model of LTPS TFT for Active Matrix Displays Addressing Circuits Simulation and Design,” vol. 51, no.3, Mar. 2004.

[15] Ilias Pappas et al, “Active-Matrix Liquid Crystal Displays - Operation, Electronics and Analog Circuits Design,” Physics Department Aristotle University of Thessaloniki Thessaloniki, Greece, [online document], www.intechweb.org [Accessed: Oct, 2013].

[16] Avago, Appl. Note 1216, pp.1-15.

[17] Xiaojun Guo et al, “Investigation on the Current Nonuniformity in Current-Mode TFT Active-Matrix Display Pixel Circuitry,” vol. 52, no.11, Nov. 2005.

[18] Wouter F. Aerts et al, “Design of an Organic Pixel Addressing Circuit for an Active-Matrix OLED Display,” vol. 49, no.12, Dec 2002.

[19] M.J. Booysen et al, “PROOF OF CONCEPT: LARGE-SCALE MONITOR AND CONTROL OF HOUSEHOLD WATER HEATING IN NEAR REAL-TIME,” International Conference on Applied Energy presented on the 1st of July 2013

[20] Arduino, “Mega 2560,” [Date Unknown]

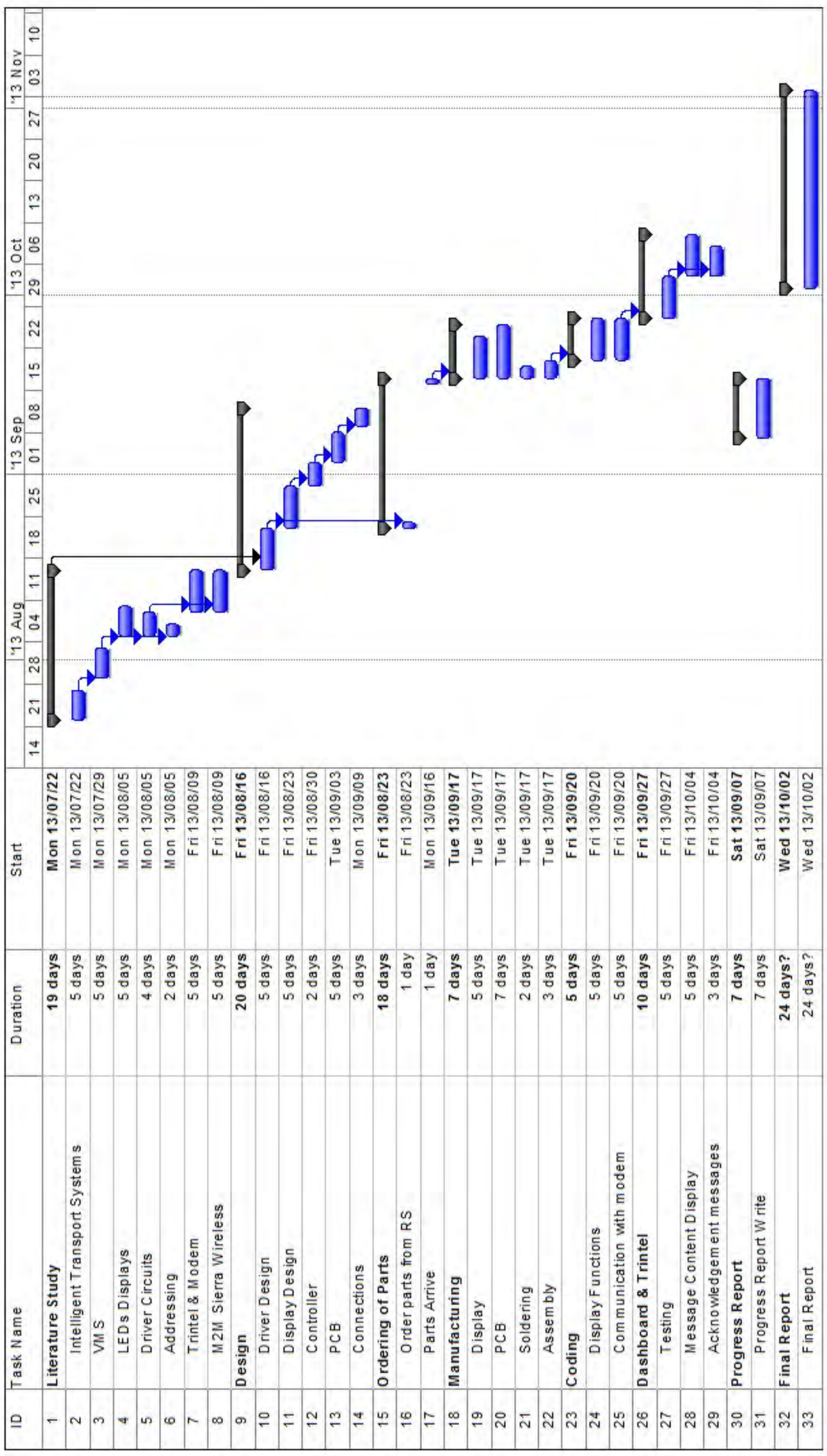
[21] MICREL, “8-Bit Serial-Input Latched Source Driver,” MIC5891 [May 2006]

[22] Texas Instruments, “16-CHANNELCONSTANT-CURRENTLEDSINKDRIVERS,” TLC5926 [July 2008]

[23] MAXIM, “RS-232 Transceivers for Multidrop Applications,” MAX3322EEUP [2003]

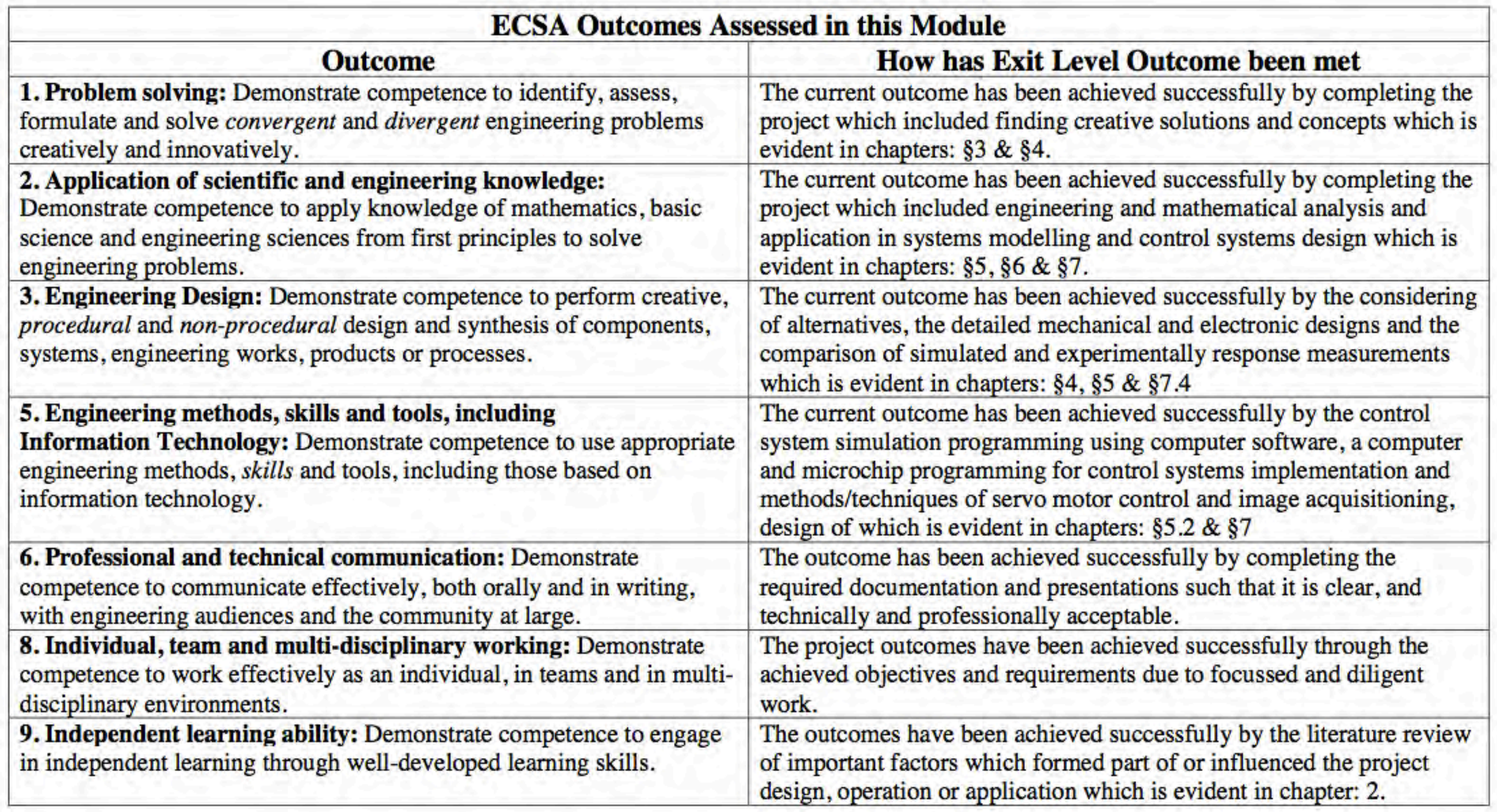
[24] Sierra Wireless, “AT Commands Interface Guide for firmware 7.46,” Interface Guide Series Report. WM_DEV_OAT_UGD, August 2011.

Appendix A : Project Management



ID	Task Name	Duration	Start
1	**Literature Study**	**19 days**	**Mon 13/07/22**
2	Intelligent Transport Systems	5 days	Mon 13/07/22
3	VMS	5 days	Mon 13/07/29
4	LEDs Displays	5 days	Mon 13/08/05
5	Driver Circuits	4 days	Mon 13/08/05
6	Addressing	2 days	Mon 13/08/05
7	Trintel & Modem	5 days	Fri 13/08/09
8	M2M Sierra Wireless	5 days	Fri 13/08/09
9	**Design**	**20 days**	**Fri 13/08/16**
10	Driver Design	5 days	Fri 13/08/16
11	Display Design	5 days	Fri 13/08/23
12	Controller	2 days	Fri 13/08/30
13	PCB	5 days	Tue 13/09/03
14	Connections	3 days	Mon 13/09/09
15	**Ordering of Parts**	**18 days**	**Fri 13/08/23**
16	Order parts from RS	1 day	Fri 13/08/23
17	Parts Arrive	1 day	Mon 13/09/16
18	**Manufacturing**	**7 days**	**Tue 13/09/17**
19	Display	5 days	Tue 13/09/17
20	PCB	7 days	Tue 13/09/17
21	Soldering	2 days	Tue 13/09/17
22	Assembly	3 days	Tue 13/09/17
23	**Coding**	**5 days**	**Fri 13/09/20**
24	Display Functions	5 days	Fri 13/09/20
25	Communication with modem	5 days	Fri 13/09/20
26	**Dashboard & Trintel**	**10 days**	**Fri 13/09/27**
27	Testing	5 days	Fri 13/09/27
28	Message Content Display	5 days	Fri 13/10/04
29	Acknowledgement messages	3 days	Fri 13/10/04
30	**Progress Report**	**7 days**	**Sat 13/09/07**
31	Progress Report Write	7 days	Sat 13/09/07
32	**Final Report**	**24 days?**	**Wed 13/10/02**
33	Final Report	24 days?	Wed 13/10/02

Appendix B : ECSA Outcome



ECSA Outcomes Assessed in this Module	
Outcome	**How has Exit Level Outcome been met**
1. Problem solving: Demonstrate competence to identify, assess, formulate and solve *convergent* and *divergent* engineering problems creatively and innovatively.	The current outcome has been achieved successfully by completing the project which included finding creative solutions and concepts which is evident in chapters: §3 & §4.
2. Application of scientific and engineering knowledge: Demonstrate competence to apply knowledge of mathematics, basic science and engineering sciences from first principles to solve engineering problems.	The current outcome has been achieved successfully by completing the project which included engineering and mathematical analysis and application in systems modelling and control systems design which is evident in chapters: §5, §6 & §7.
3. Engineering Design: Demonstrate competence to perform creative, *procedural* and *non-procedural* design and synthesis of components, systems, engineering works, products or processes.	The current outcome has been achieved successfully by the considering of alternatives, the detailed mechanical and electronic designs and the comparison of simulated and experimentally response measurements which is evident in chapters: §4, §5 & §7.4
5. Engineering methods, skills and tools, including Information Technology: Demonstrate competence to use appropriate engineering methods, *skills* and tools, including those based on information technology.	The current outcome has been achieved successfully by the control system simulation programming using computer software, a computer and microchip programming for control systems implementation and methods/techniques of servo motor control and image acquisitioning, design of which is evident in chapters: §5.2 & §7
6. Professional and technical communication: Demonstrate competence to communicate effectively, both orally and in writing, with engineering audiences and the community at large.	The outcome has been achieved successfully by completing the required documentation and presentations such that it is clear, and technically and professionally acceptable.
8. Individual, team and multi-disciplinary working: Demonstrate competence to work effectively as an individual, in teams and in multi-disciplinary environments.	The project outcomes have been achieved successfully through the achieved objectives and requirements due to focussed and diligent work.
9. Independent learning ability: Demonstrate competence to engage in independent learning through well-developed learning skills.	The outcomes have been achieved successfully by the literature review of important factors which formed part of or influenced the project design, operation or application which is evident in chapter: 2.

Appendix C : Software

Flow Diagrams of Functions:

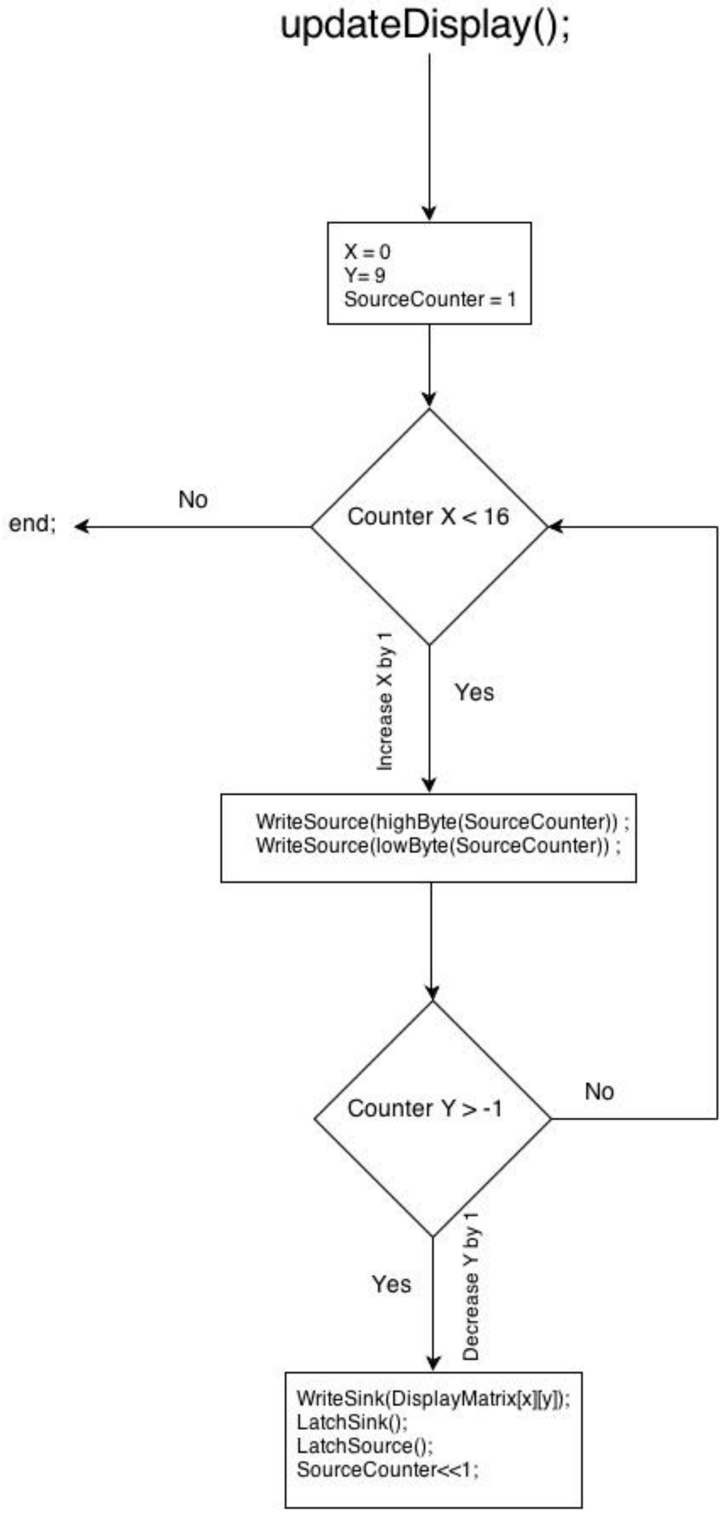


Figure 61: updateDisplay(); Flow Diagram

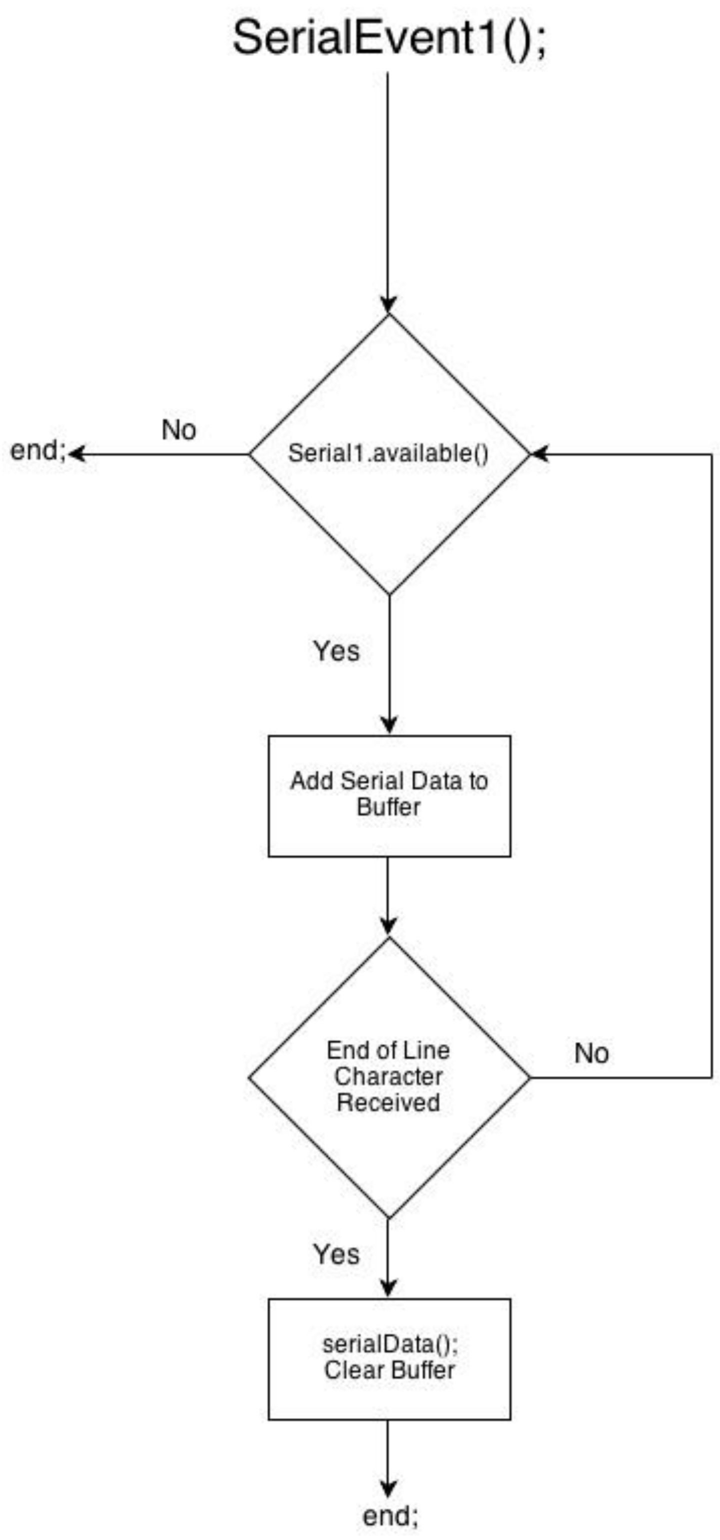

SerialEvent1();
Serial1.available()
No
end;
Yes
Add Serial Data to Buffer
End of Line Character Received
No
Yes
serialData(); Clear Buffer
end;

Figure 62: SerialEvent1(); Flow Diagram

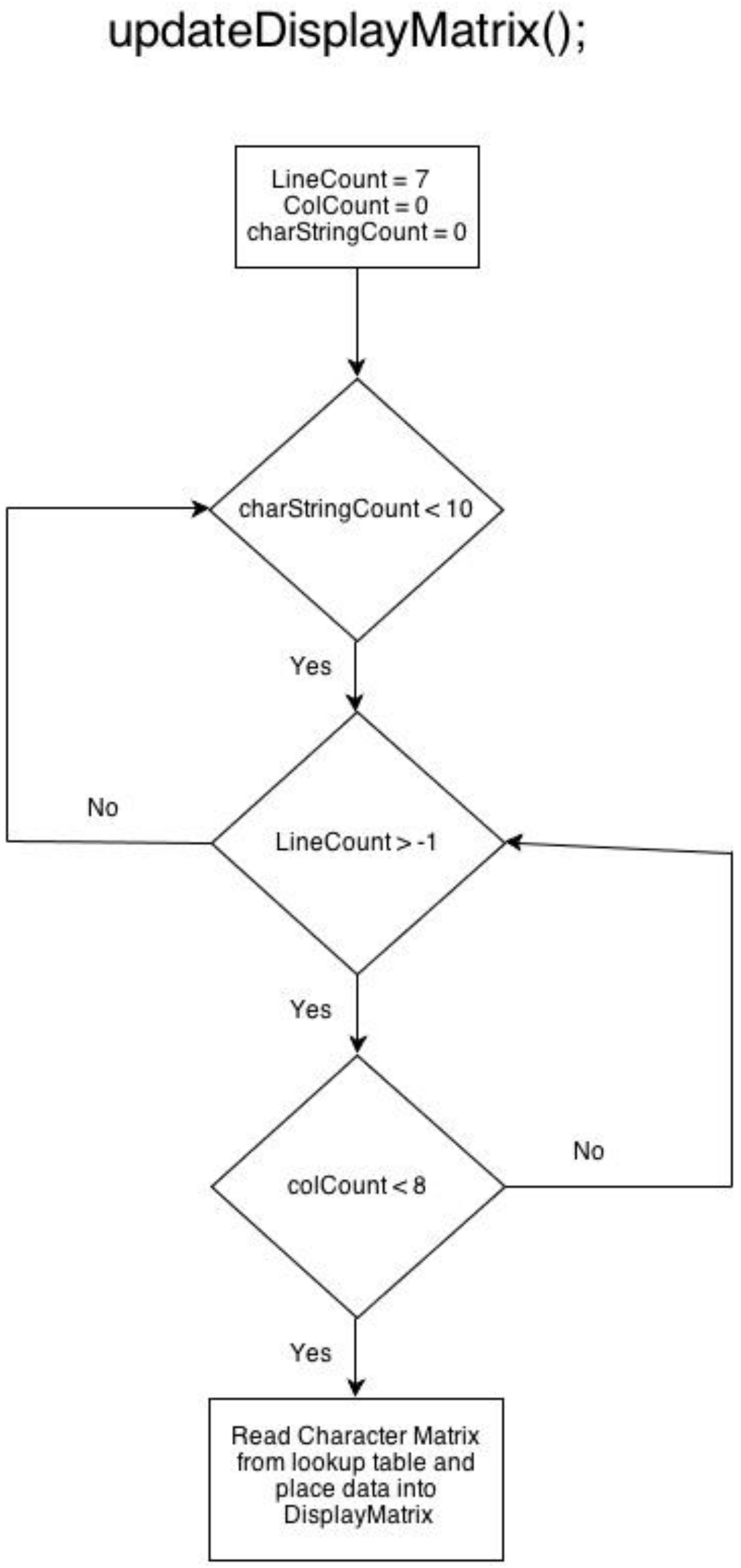


Figure 63: updateDisplayMatrix(); Flow Diagram

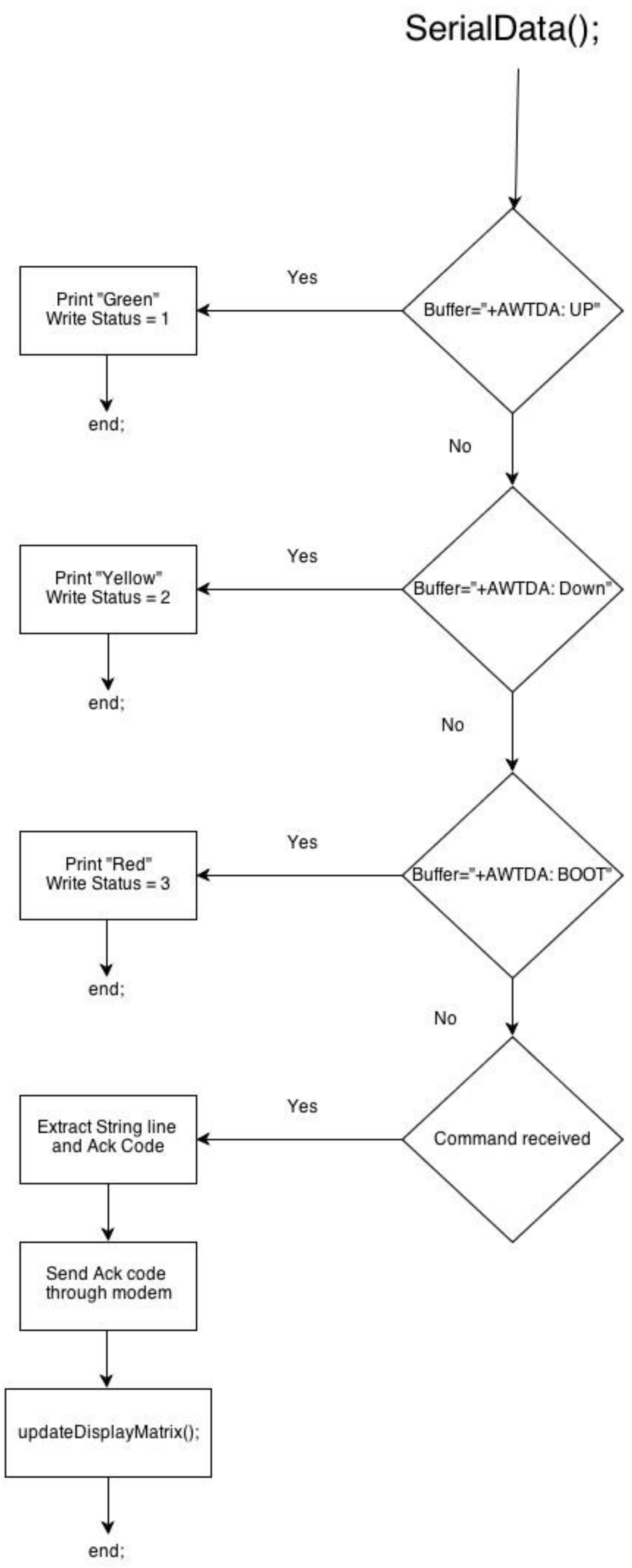


Figure 64: SerialData(); Flow Diagram

Appendix D : Circuit Construction

Gerber files:

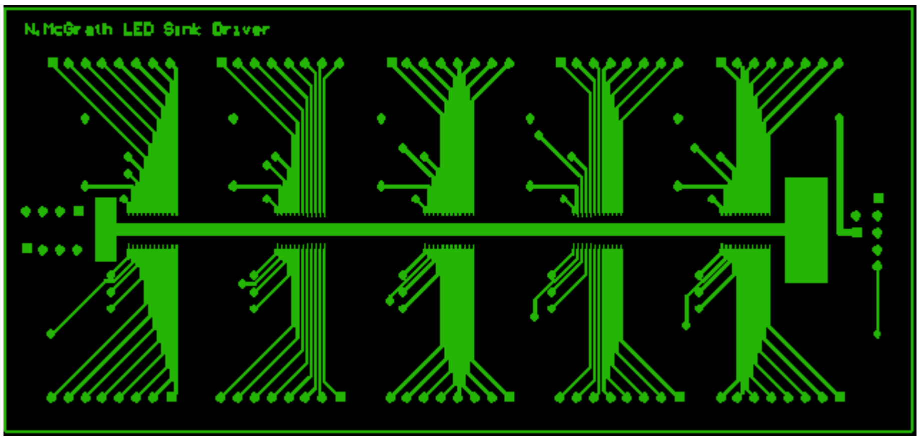


Figure 65: LED Constant Current Sink Driver Gerber File top

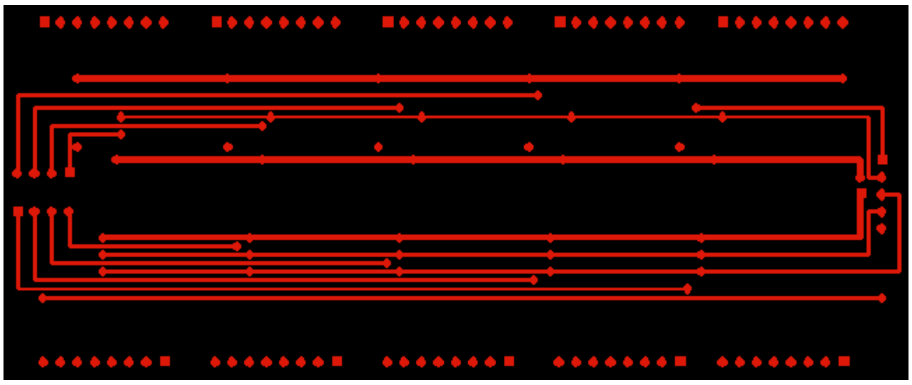


Figure 66: LED Constant Current Driver Gerber File Bottom

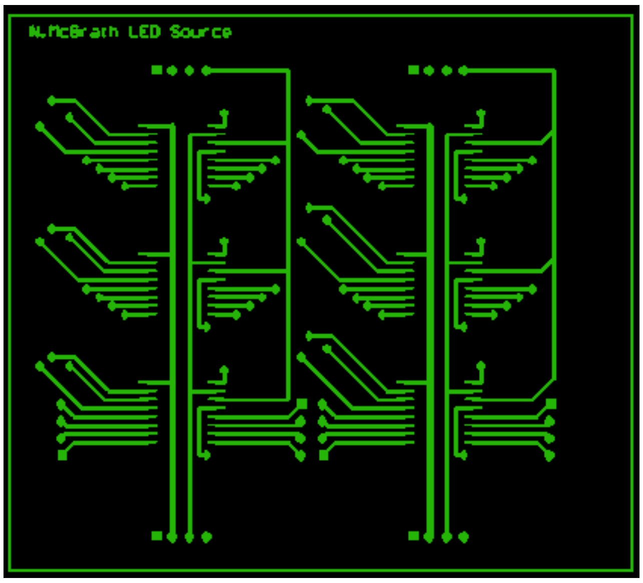


Figure 67: LED Voltage Source Driver Gerber File Top

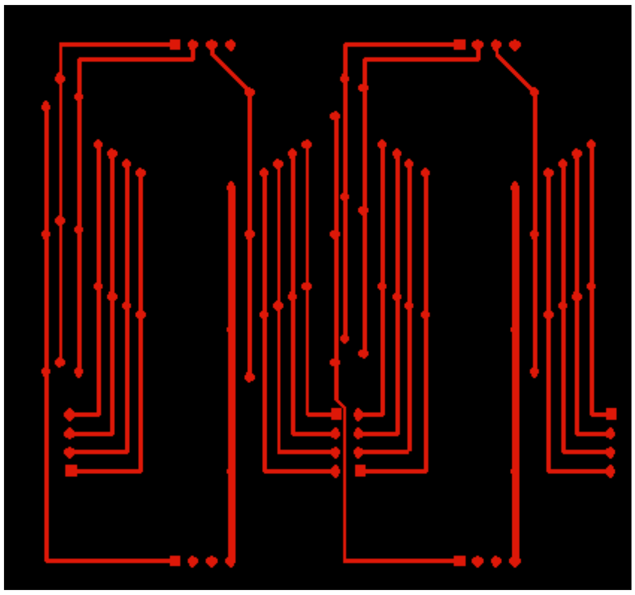


Figure 68: Voltage Source Driver Gerber File Bottom

Physical Circuit:

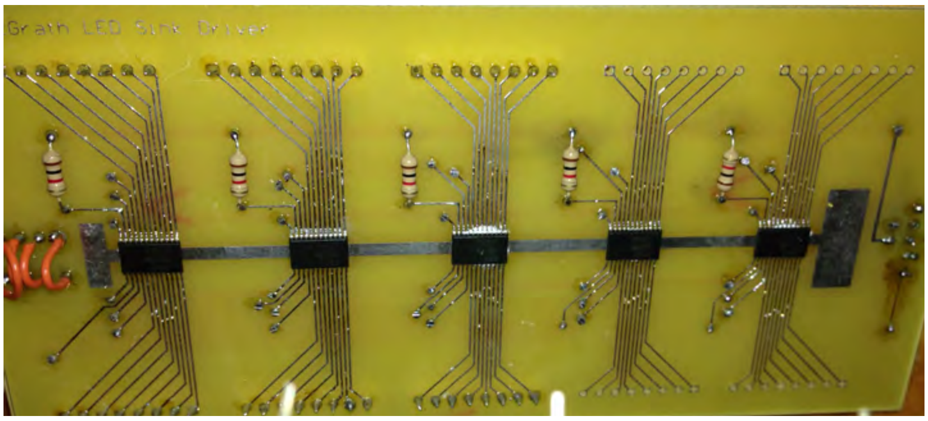


Figure 69: LED Sink Driver

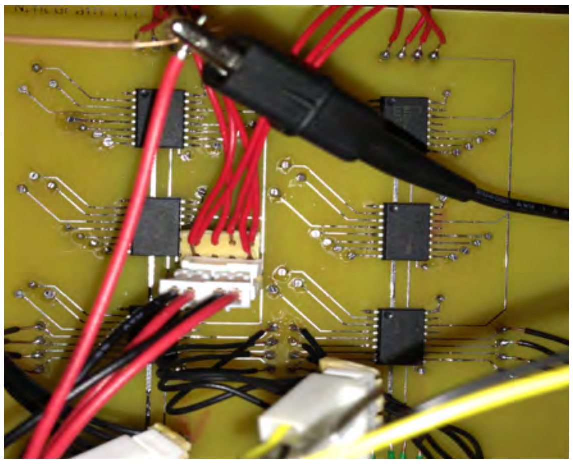


Figure 70: LED Voltage Source Driver

System Diagram:

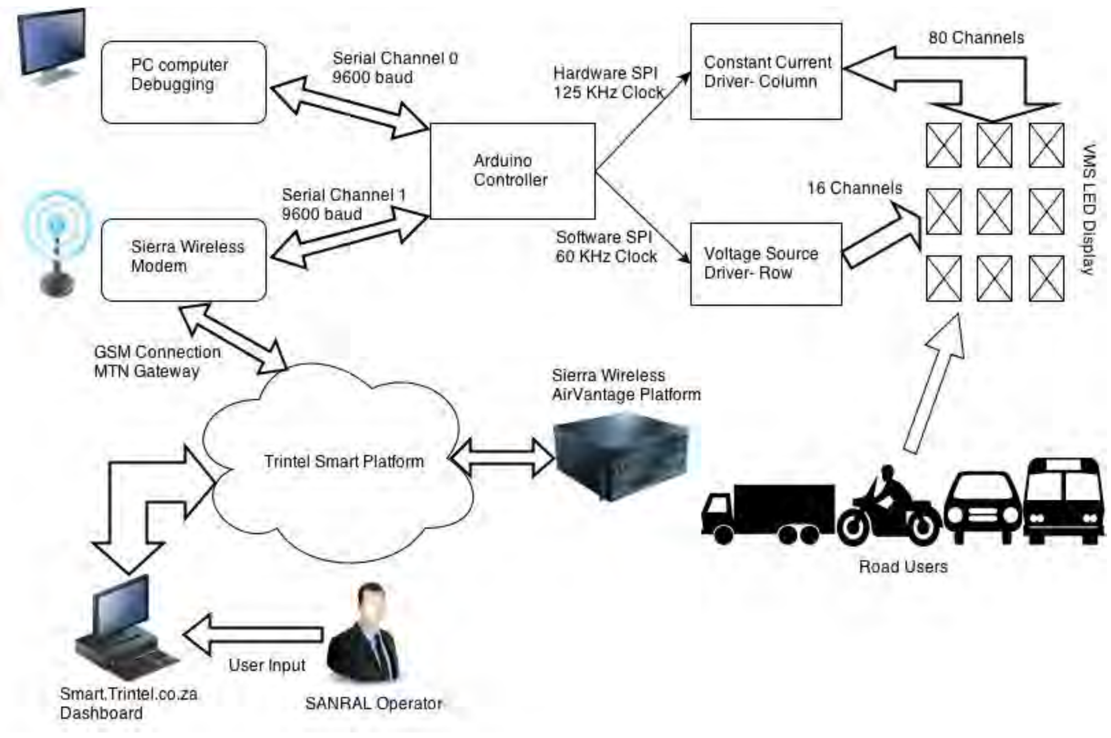


Figure 71: Full System Diagram

Circuit Diagrams:

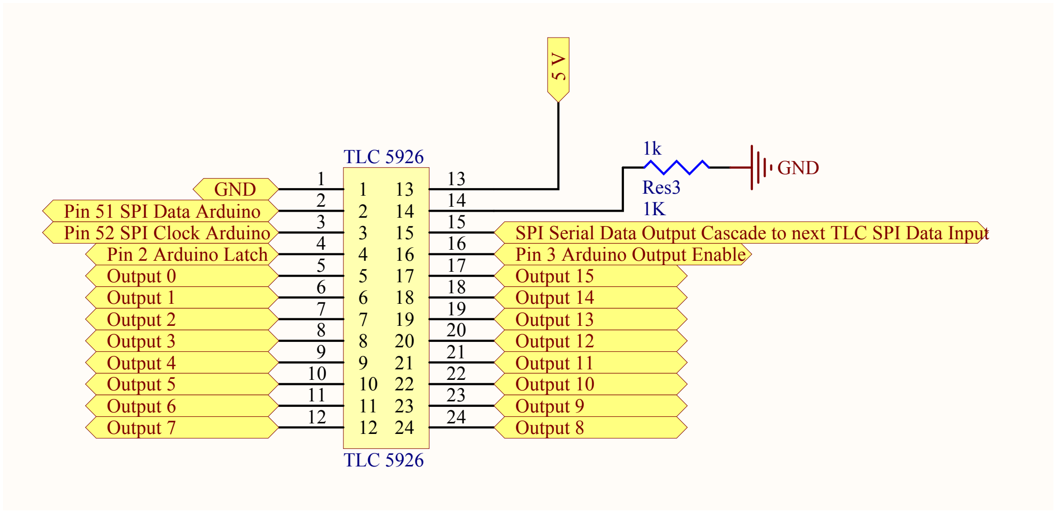


Figure 72: TLC 5926 Connections

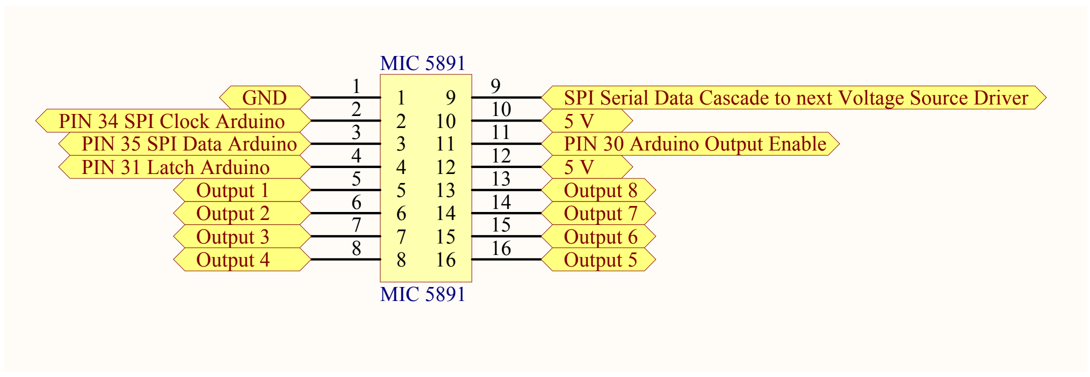


Figure 73: MIC 5891 Connections

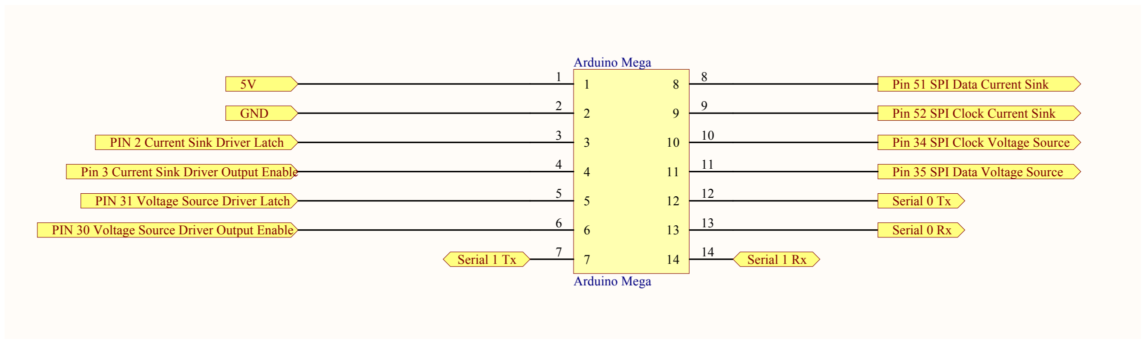

Arduino Mega
5V
GND
PIN 2 Current Sink Driver Latch
Pin 3 Current Sink Driver Output Enable
PIN 31 Voltage Source Driver Latch
PIN 30 Voltage Source Driver Output Enable
Serial 1 Tx
Pin 51 SPI Data Current Sink
Pin 52 SPI Clock Current Sink
Pin 34 SPI Clock Voltage Source
Pin 35 SPI Data Voltage Source
Serial 0 Tx
Serial 0 Rx
Serial 1 Rx
Arduino Mega

Figure 74: Arduino Connections

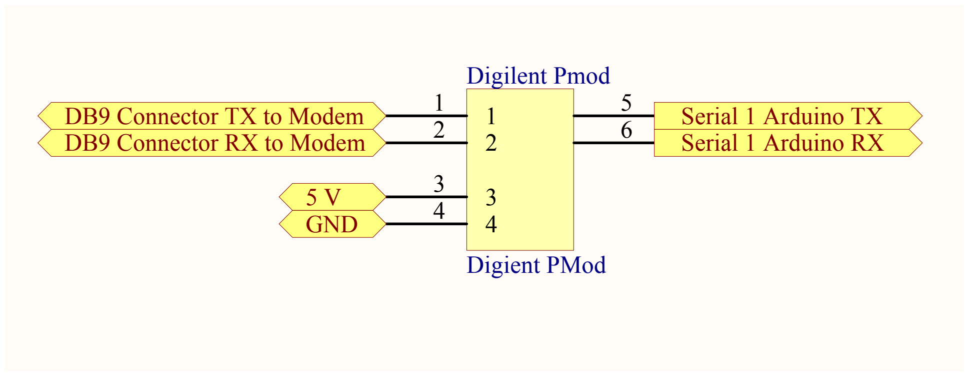


Figure 75: Digilent PMod Serial Voltage Converter Connections

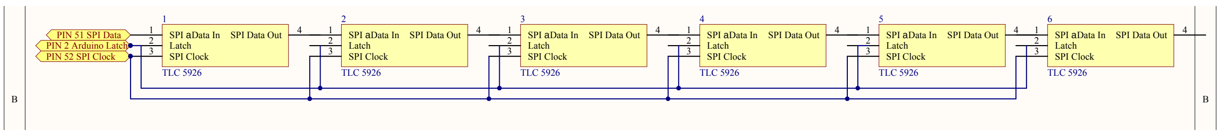


Figure 76: TLC 5926 Communication Connections

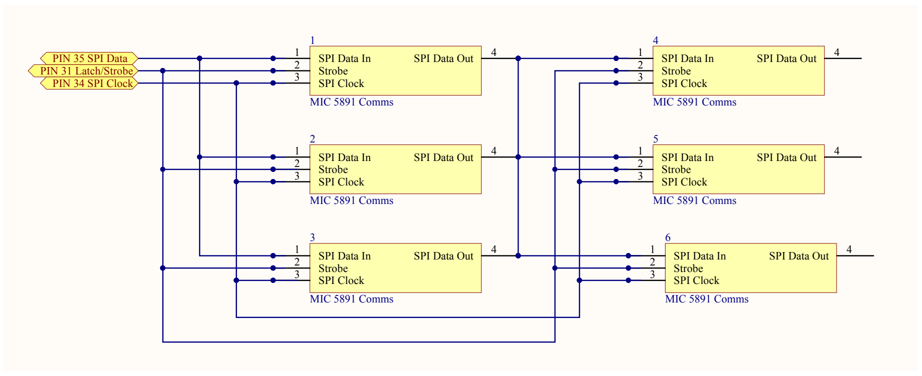


Figure 77: MIC 5891 Communication Connections

Appendix E : Code

```
// Neethling McGrath 15707059
//

#include <SPI.h>
#include <Timer.h>

Timer t ;


//Global Variables==============================================================

 int State ;
 String buffer ;



 //Flags

 int AllowPrint= 0 ;

 byte DisplayMatrix[16][10];

String CharString1 = "         ";
String CharString2 = "         ";

                         const char LUT[128][8] =
                         {
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
                                     {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //

          {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, //
    {0x00,0x60,0xfa,0xfa,0x60,0x00,0x00,0x00}, // !
    {0x00,0xe0,0xe0,0x00,0xe0,0xe0,0x00,0x00}, // "
    {0x28,0xfe,0xfe,0x28,0xfe,0xfe,0x28,0x00}, // #
    {0x24,0x74,0xd6,0xd6,0x5c,0x48,0x00,0x00}, // $
    {0x62,0x66,0x0c,0x18,0x30,0x66,0x46,0x00}, // %
    {0x0c,0x5e,0xf2,0xba,0xec,0x5e,0x12,0x00}, // &
```

```
{0x20,0xe0,0xc0,0x00,0x00,0x00,0x00,0x00}, // '
{0x00,0x38,0x7c,0xc6,0x82,0x00,0x00,0x00}, // (
{0x00,0x82,0xc6,0x7c,0x38,0x00,0x00,0x00}, // )
{0x10,0x54,0x7c,0x38,0x38,0x7c,0x54,0x10}, // *
{0x10,0x10,0x7c,0x7c,0x10,0x10,0x00,0x00}, // +
{0x00,0x05,0x07,0x06,0x00,0x00,0x00,0x00}, // ,
{0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00}, // -
{0x00,0x00,0x06,0x06,0x00,0x00,0x00,0x00}, // .
{0x06,0x0c,0x18,0x30,0x60,0xc0,0x80,0x00}, // /
{0x7c,0xfe,0x9a,0xb2,0xfe,0x7c,0x00,0x00}, // 0
{0x42,0x42,0xfe,0xfe,0x02,0x02,0x00,0x00}, // 1
{0x46,0xce,0x9a,0x92,0xf6,0x66,0x00,0x00}, // 2
{0x44,0xc6,0x92,0x92,0xfe,0x6c,0x00,0x00}, // 3
{0x18,0x38,0x68,0xc8,0xfe,0xfe,0x08,0x00}, // 4
{0xe4,0xe6,0xa2,0xa2,0xbe,0x9c,0x00,0x00}, // 5
{0x3c,0x7e,0xd2,0x92,0x9e,0x0c,0x00,0x00}, // 6
{0xc0,0xc6,0x8e,0x98,0xf0,0xe0,0x00,0x00}, // 7
{0x6c,0xfe,0x92,0x92,0xfe,0x6c,0x00,0x00}, // 8
{0x60,0xf2,0x92,0x96,0xfc,0x78,0x00,0x00}, // 9
{0x00,0x00,0x36,0x36,0x00,0x00,0x00,0x00}, // :
{0x00,0x05,0x37,0x36,0x00,0x00,0x00,0x00}, // ;
{0x10,0x38,0x6c,0xc6,0x82,0x00,0x00,0x00}, // <
{0x28,0x28,0x28,0x28,0x28,0x28,0x00,0x00}, // =
{0x00,0x82,0xc6,0x6c,0x38,0x10,0x00,0x00}, // >
{0x40,0xc0,0x8a,0x9a,0xf0,0x60,0x00,0x00}, // ?
{0x7c,0xfe,0x82,0xba,0xba,0xf8,0x78,0x00}, // @
{0x3e,0x7e,0xc8,0xc8,0x7e,0x3e,0x00,0x00}, // A
{0x82,0xfe,0xfe,0x92,0x92,0xfe,0x6c,0x00}, // B
{0x38,0x7c,0xc6,0x82,0x82,0xc6,0x44,0x00}, // C
{0x82,0xfe,0xfe,0x82,0xc6,0xfe,0x38,0x00}, // D
{0x82,0xfe,0xfe,0x92,0xba,0x82,0xc6,0x00}, // E
{0x82,0xfe,0xfe,0x92,0xb8,0x80,0xc0,0x00}, // F
{0x38,0x7c,0xc6,0x82,0x8a,0xce,0x4e,0x00}, // G
{0xfe,0xfe,0x10,0x10,0xfe,0xfe,0x00,0x00}, // H
{0x00,0x82,0xfe,0xfe,0x82,0x00,0x00,0x00}, // I
{0x0c,0x0e,0x02,0x82,0xfe,0xfc,0x80,0x00}, // J
{0x82,0xfe,0xfe,0x10,0x38,0xee,0xc6,0x00}, // K
{0x82,0xfe,0xfe,0x82,0x02,0x06,0x0e,0x00}, // L
{0xfe,0xfe,0x60,0x30,0x60,0xfe,0xfe,0x00}, // M
{0xfe,0xfe,0x60,0x30,0x18,0xfe,0xfe,0x00}, // N
{0x38,0x7c,0xc6,0x82,0xc6,0x7c,0x38,0x00}, // O
{0x82,0xfe,0xfe,0x92,0x90,0xf0,0x60,0x00}, // P
{0x78,0xfc,0x84,0x8e,0xfe,0x7a,0x00,0x00}, // Q
{0x82,0xfe,0xfe,0x98,0x9c,0xf6,0x62,0x00}, // R
{0x64,0xe6,0xb2,0x9a,0xde,0x4c,0x00,0x00}, // S
{0xc0,0x82,0xfe,0xfe,0x82,0xc0,0x00,0x00}, // T
{0xfe,0xfe,0x02,0x02,0xfe,0xfe,0x00,0x00}, // U
{0xf8,0xfc,0x06,0x06,0xfc,0xf8,0x00,0x00}, // V
{0xfe,0xfe,0x0c,0x18,0x0c,0xfe,0xfe,0x00}, // W
{0xc6,0xee,0x38,0x10,0x38,0xee,0xc6,0x00}, // X
{0xe0,0xf2,0x1e,0x1e,0xf2,0xe0,0x00,0x00}, // Y
{0xe6,0xce,0x9a,0xb2,0xe2,0xc6,0x8e,0x00}, // Z
{0x00,0xfe,0xfe,0x82,0x82,0x00,0x00,0x00}, // [
{0x80,0xc0,0x60,0x30,0x18,0x0c,0x06,0x00}, // "\"
{0x00,0x82,0x82,0xfe,0xfe,0x00,0x00,0x00}, // ]
{0x10,0x30,0x60,0xc0,0x60,0x30,0x10,0x00}, // ^
{0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01}, // _
{0x00,0x00,0xc0,0xe0,0x20,0x00,0x00,0x00}, // `
{0x04,0x2e,0x2a,0x2a,0x3c,0x1e,0x02,0x00}, // a
{0x82,0xfc,0xfe,0x22,0x22,0x3e,0x1c,0x00}, // b
{0x1c,0x3e,0x22,0x22,0x36,0x14,0x00,0x00}, // c
{0x0c,0x1e,0x12,0x92,0xfc,0xfe,0x02,0x00}, // d
{0x1c,0x3e,0x2a,0x2a,0x3a,0x18,0x00,0x00}, // e
{0x12,0x7e,0xfe,0x92,0xc0,0x40,0x00,0x00}, // f
{0x19,0x3d,0x25,0x25,0x1f,0x3e,0x20,0x00}, // g
{0x82,0xfe,0xfe,0x10,0x20,0x3e,0x1e,0x00}, // h
{0x00,0x22,0xbe,0xbe,0x02,0x00,0x00,0x00}, // i
{0x02,0x23,0x21,0xbf,0xbe,0x00,0x00,0x00}, // j
{0x82,0xfe,0xfe,0x08,0x1c,0x36,0x22,0x00}, // k
{0x00,0x82,0xfe,0xfe,0x02,0x00,0x00,0x00}, // l
{0x3e,0x3e,0x30,0x18,0x30,0x3e,0x1e,0x00}, // m
{0x3e,0x3e,0x20,0x20,0x3e,0x1e,0x00,0x00}, // n
{0x1c,0x3e,0x22,0x22,0x3e,0x1c,0x00,0x00}, // o
{0x21,0x3f,0x1f,0x25,0x24,0x3c,0x18,0x00}, // p
```

```
        {0x18,0x3c,0x24,0x25,0x1f,0x3f,0x21,0x00}, // q
        {0x22,0x3e,0x1e,0x22,0x38,0x18,0x00,0x00}, // r
        {0x12,0x3a,0x2a,0x2a,0x2e,0x24,0x00,0x00}, // s
        {0x00,0x20,0x7c,0xfe,0x22,0x24,0x00,0x00}, // t
        {0x3c,0x3e,0x02,0x02,0x3c,0x3e,0x02,0x00}, // u
        {0x38,0x3c,0x06,0x06,0x3c,0x38,0x00,0x00}, // v
        {0x3c,0x3e,0x06,0x0c,0x06,0x3e,0x3c,0x00}, // w
        {0x22,0x36,0x1c,0x08,0x1c,0x36,0x22,0x00}, // x
        {0x39,0x3d,0x05,0x05,0x3f,0x3e,0x00,0x00}, // y
        {0x32,0x26,0x2e,0x3a,0x32,0x26,0x00,0x00}, // z
        {0x10,0x10,0x7c,0xee,0x82,0x82,0x00,0x00}, // {
        {0x00,0x00,0x00,0xee,0xee,0x00,0x00,0x00}, // |
        {0x82,0x82,0xee,0x7c,0x10,0x10,0x00,0x00}, // }
        {0x40,0xc0,0x80,0xc0,0x40,0xc0,0x80,0x00}, // ~
        {0x1e,0x3e,0x62,0xc2,0x62,0x3e,0x1e,0x00}, //
};


//Global Variables===========================================================

void LatchSource()
{

  digitalWrite(31, HIGH);

  digitalWrite(31, LOW);
}

void LatchSink()
{
  digitalWrite(2, HIGH);
  digitalWrite(2, LOW);
}

void WriteSource(byte message)
{
  // digitalWrite(35, HIGH);
   shiftOut(35, 34, MSBFIRST, message);
   //digitalWrite(35, HIGH);

  // LatchSource();
}

void WriteSink(byte message)
{

   SPI.transfer(message) ;
   //LatchSource() ;
  // LatchSink();
}

//Display update Function=====================================================



void updateDisplayMatrix()
{


  int currentBit = 0;
  byte constructByte = 0 ;

//Serial.println("Begin") ;


  for (int charStringCount = 0 ; charStringCount < 10 ; charStringCount++)
  {

      for(int lineCount = 7 ; lineCount >= 0 ; lineCount--)
      {


        //Serial.println(Test,BIN);
        constructByte = 0 ;
```

```
         for(int colCount = 0 ; colCount < 8  ; colCount++)
         {
          //Test =  LUT[CharString1[charStringCount]][lineCount] ;
          currentBit = bitRead(LUT[CharString1[charStringCount]][colCount],lineCount) ;
          //Serial.print(currentBit) ;

          if (currentBit == 0)

          {
           bitWrite(constructByte,colCount,LOW) ;
          }
          else if (currentBit == 1)
          {
           bitWrite(constructByte,colCount,HIGH) ;
          }



       }

       DisplayMatrix[7-lineCount][charStringCount] = constructByte ;
         //Serial.println() ;

      }
 //Serial.println("End") ;
  }



 //=========================================================================================
 ============

 for (int charStringCount = 0 ; charStringCount < 10 ; charStringCount++)
  {

      for(int lineCount = 7 ; lineCount >= 0 ; lineCount--)
      {



        constructByte = 0 ;
         for(int colCount = 0 ; colCount < 8  ; colCount++)
         {

          currentBit = bitRead(LUT[CharString2[charStringCount]][colCount],lineCount) ;
          //Serial.print(currentBit) ;

          if (currentBit == 0)

          {
           bitWrite(constructByte,colCount,LOW) ;
          }
          else if (currentBit == 1)
          {
           bitWrite(constructByte,colCount,HIGH) ;
          }



       }

       DisplayMatrix[15-lineCount][charStringCount] = constructByte ;
         //Serial.println() ;

      }
 //Serial.println("End") ;
  }





 }
```

```
void serialDisplayout()
{

 for(int x = 0 ; x < 16 ; x++)
  {

    for(int y = 0 ; y < 10 ; y++)
    {

      for(int lineCount = 0 ; lineCount < 8 ; lineCount++)
      {
      Serial.print(bitRead(DisplayMatrix[x][y],lineCount));
      }



    }
   Serial.println();

  }


}

void updateDisplay()
{




  int SourceCounter = 1 ;

for (int x = 0 ; x < 16 ; x++)

{

  WriteSource(highByte(SourceCounter)) ;
  WriteSource(lowByte(SourceCounter)) ;

  for (int y = 9 ; y > -1  ; y--)
  {
  WriteSink(DisplayMatrix[x][y]);

  }
 //Serial.println() ;
 LatchSink();

 LatchSource();
 // Serial.println(SourceCounter,BIN);
  SourceCounter = SourceCounter << 1 ;

}
}


//Serial Data Format==================================================================

void serialData()
{



  if (buffer.charAt(8) == 'U')
 {


   Serial.println("Status:Green") ;
   State = 1;
   Serial1.print("AT+AWTDA=d,\"VMS\",1,\"Status,INT32,1");
   Serial1.write(13);
```

```
}

  if (buffer.charAt(8) == 'B')
{

  Serial.println("Status:Red") ;

  State = 3 ;
  Serial1.print("AT+AWTDA=d,\"VMS\",1,\"Status,INT32,3");
 Serial1.write(13);

}

   if (buffer.charAt(8) == 'D')
{

  Serial.println("Status:Yellow") ;

  State = 2;
  Serial1.print("AT+AWTDA=d,\"VMS\",1,\"Status,INT32,2");
   Serial1.write(13);

}

  if (buffer.charAt(8) == 'c')
{



  if (buffer.charAt(34) == '1')
  {
   // Serial.println("ASAS");
    String line ;
    String Ack ;
    int x = 40 ;

    do
    {
    line += buffer.charAt(x);
    x++ ;
    }
    while (buffer.charAt(x) != ',');

    Serial.println(line) ;
    CharString1 = line ;
    x = 10 ;

     do
    {
    Ack += buffer.charAt(x);
    x++ ;
    }
    while (buffer.charAt(x) != ',');

    Serial.println(Ack) ;

    Serial1.write("AT+AWTDA=a,\"VMS\",");

      for (int x = 0 ; x < Ack.length() ; x++)
      {
       Serial1.write(Ack.charAt(x));

      }
       Serial1.write(13) ;



  }

   if (buffer.charAt(34) == '2')
  {
   // Serial.println("ASAS");
```

```
    String line ;
    String Ack ;
    int x = 40 ;

    do
    {
    line += buffer.charAt(x);
    x++ ;
    }
    while (buffer.charAt(x) != ',');

    Serial.println(line) ;

    CharString2 = line ;


    x = 10 ;



     do
    {
    Ack += buffer.charAt(x);
    x++ ;
    }
    while (buffer.charAt(x) != ',');

    Serial.println(Ack) ;

    Serial1.write("AT+AWTDA=a,\"VMS\",");

      for (int x = 0 ; x < Ack.length() ; x++)
      {
       Serial1.write(Ack.charAt(x));

      }
       Serial1.write(13) ;


  }

  updateDisplayMatrix() ;

 }





}

//Serial Data Format==================================================================


void setup()
{

 //Pins SETUP==================================================================

 //Sink Pins

 pinMode(2, OUTPUT);//XLAT-Sink
 pinMode(3, OUTPUT);//output enable-Sink

 pinMode(51, OUTPUT);//MOSI DATA-Sink-SPI
 pinMode(52, OUTPUT);//SPI Clock-Sink-SPI

 //TEST PINS

 pinMode(8,OUTPUT) ;
 pinMode(9,OUTPUT) ;
```

```
  digitalWrite(8,LOW) ;

  //Source Pins

  pinMode(31, OUTPUT);//XLAT-Source
  pinMode(30, OUTPUT);//output enable-Source

  pinMode(34, OUTPUT);//SPI Clock-Source
  pinMode(35, OUTPUT);//MOSI DATA-Source

//Pins SETUP=================================================================


//Coms SETUP=================================================================

  //SPI

  SPI.setBitOrder(MSBFIRST);//Most Significant Bit First
  SPI.setDataMode(SPI_MODE0);// Mode 0 Rising edge of data, keep clock low
  SPI.setClockDivider(SPI_CLOCK_DIV128);//Run the data in at 16MHz/128 - 125 KHz
  SPI.begin() ;

  //Serial

  Serial.begin(9600) ;
  Serial1.begin(9600);


//Set Pins===================================================================

   digitalWrite(2, LOW);  //set Latch-Sink low
   digitalWrite(3, LOW);  //set OE-Sink low
   digitalWrite(31, LOW);  //set Latch-Source low
   digitalWrite(30, LOW);  //set OE-Source low

//Init All registers=========================================================



  for (int x = 0 ; x < 16 ; x++)
{
  for (int y = 0 ; y < 10 ; y++)
  {
  DisplayMatrix[x][y] = 0 ;
  }
}

  for (int SourceDrivers = 0; SourceDrivers < 2 ; SourceDrivers++)
  {

      WriteSource(0x00);



  }

    for (int SinkDrivers = 0; SinkDrivers < 5 ; SinkDrivers++)
  {

      WriteSink(0x00);

  }

  updateDisplayMatrix();
 // serialDisplayout() ;
//Clear All Registers========================================================


//Setup Commands=============================================================

  Serial1.print("AT+AWTDA=c,\"VMS\",\"UpdateString1\"");
  Serial1.write(13);

  delay(100) ;
 Serial1.print("AT+AWTDA=c,\"VMS\",\"UpdateString2\"");
 Serial1.write(13);




//Setup Timer=================================================================

int rowpos = 1 ;
byte Hbyterow ;
byte Lbyterow ;

int colpos = 1 ;
byte Hbytecol ;
byte Lbytecol ;

 for (int Row = 0 ; Row<10 ; Row++)
 {

    Hbyterow = highByte(rowpos);
    Lbyterow = lowByte(rowpos);

    WriteSource(Hbyterow);
    WriteSource(Lbyterow);

    rowpos = rowpos << 1 ;

    colpos = 1 ;

    for (int Col = 0 ; Col<14 ; Col++)
 {
    Hbytecol = highByte(colpos);
    Lbytecol = lowByte(colpos);

    WriteSink(Hbytecol);

    WriteSink(Lbytecol);

    colpos = colpos << 1 ;

    delay(0);
}

}

    updateDisplayMatrix();
  //  serialDisplayout() ;
}

int LedState = LOW ;

void loop()
{


 updateDisplay();


  if (LedState == LOW)
    LedState = HIGH;
   else
    LedState = LOW;

    digitalWrite(8,LedState) ;


}

void serialEvent1() {
 char temp ;
 int stringEnd ;

 stringEnd = 0 ;

  while (Serial1.available()) {



     temp = (char)Serial1.read();
     buffer += temp;

    // Serial.print(temp,DEC) ;
    // Serial.print(temp) ;

     if(int(temp) == 10)

     {
      stringEnd = 1;
      break ;
     }
   }

if (stringEnd == 1)
{



 Serial.print(buffer) ;

 serialData();

 buffer = "" ;



}



}
```