Bilkent University Department of Electrical and Electronics Engineering EEE – 313

Name of the student: Hasaan Ijaz (21801203)

Term project: Bluetooth Controlled RGB LED Strips

Table of contents

Abstract 3

Introduction 3

Materials and Methods 4

Bluetooth Module (HC-05) 4

Arduino Nano 4

RGB LED strip (SMD 5050) 7

NMOS Transistor 9

Arduino Output 10

Schematic 23

Hardware 25

Conclusion 35

References 38

Appendix 39

Fading LEDs 39

Bluetooth Communication 40

Abstract

As the Internet of Things (IOT) encapsulates all aspects of human life and empowers users to manipulate technology at the tip of their fingers, IOT has had a profound impact on electronic circuit design. Failure to keep up with the new status quo signals possible bankruptcy for circuit design companies. Therefore, for my project I create an RGB LED strip which can be operated using a mobile application. The project is realized by integrating the knowledge of transistors, and microcontrollers.

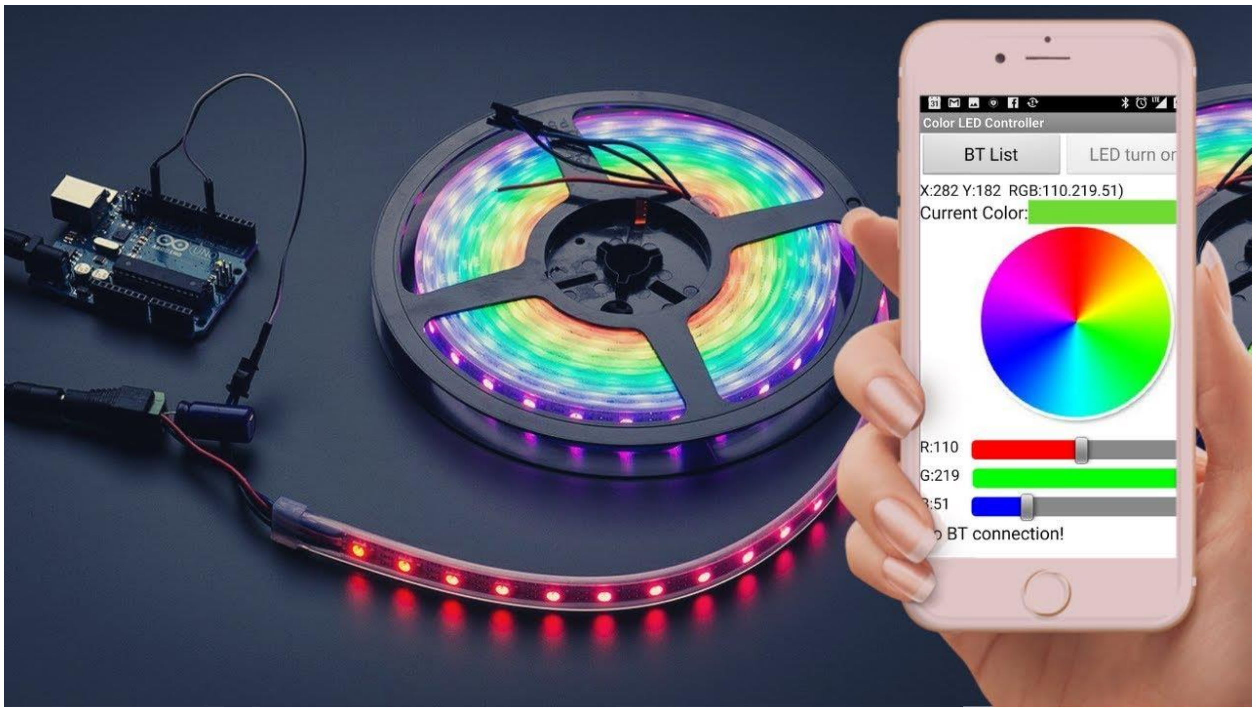


Figure 1: Image showing visual representation setup of the ECD project
Source: Copied from [1]

Introduction

This project is an attempt to exploit NMOS transistor in active (saturation) region to achieve a state-of-the art Bluetooth controlled LED light system. To control the RGB LED strip using a mobile application, a Bluetooth module is required. For the project, HC-05 Bluetooth module is used because it is cheap and easy-to-use. This Bluetooth module is only capable of reading and sending data to a microcontroller. Therefore, Arduino Nano – a microcontroller – was incorporated into the project. The microcontroller processes information fed by the Bluetooth module and reacts to it by generating a corresponding voltage level at the PMW pins. The higher the PMW pin value, higher the voltage generated across the digital pin. For higher voltages generated at the microcontroller, we expect the brightness of the LED strip to increase. However, the SMD 5050 is designed such that it is brighter at low voltage value at the RGB channels. Therefore, to invert the voltage values and get desired outputs (brightness of the strip), we use a MOSFET as a switch. It is possible to achieve the same functionality by programming the microcontroller to account for SMD 5050 behavior. For the project's purposes, however, a MOSFET in active region is used.

Materials and Methods

Bluetooth Module (HC-05)

HC-05 is a full-duplex device which uses Serial Port Protocol for communication purposes (Figure 2). The device is capable to handle baud rates – rate at which information is sent and received – of 9600, 19200, 38400, etc. A higher baud rate would mean communication is faster so more bits can be sent per second. For successful communication with the device, understanding of HC-05 pins is vital. The TX pin is responsible for transmitting serial data that has been received using Bluetooth paired device. Similarly, data sent to the Bluetooth module shows up at the RX pin which is then shared with the paired device. There is an on-chip LED as well which shows the status of the module:

1. If the LED is blinking repeatedly, then the device is awaiting connection.
2. If the LED blinks twice in a second, then the device has been successfully paired up
3. If the LED blinks once in two seconds, then the device is in command mode and is paired up.

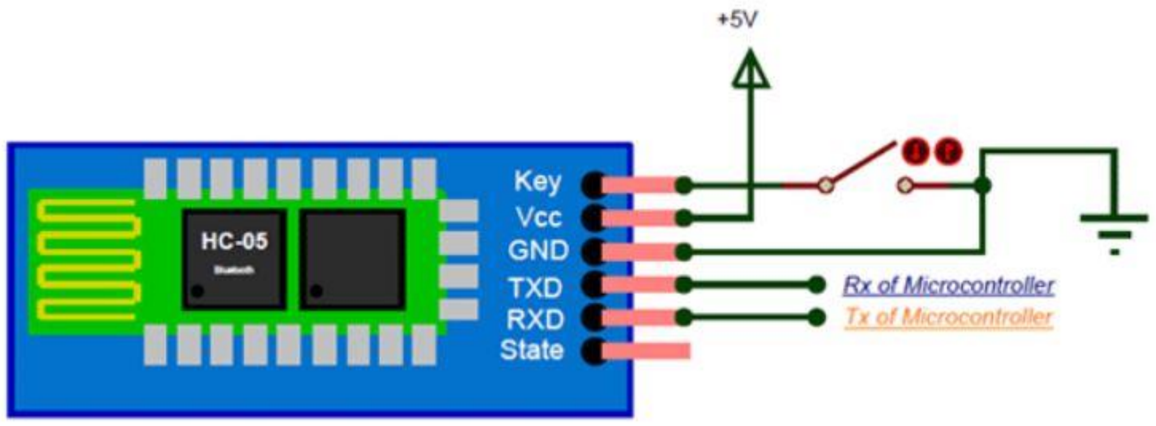


Figure 2: Image showing an animated schematic of HC-05 with its pins labelled. This device is responsible for communicating the mobile application data with the Arduino nano board. Source: Copied from [5]

The value given out will be read and fed to the microcontroller which will then use built-in program in the Atmel processor to determine the RGB values. The calculated output will then be displayed on output pins of the Arduino nano and fed to the RBG LED strips.

Arduino Nano

Arduino Nano is a microcontroller that uses the ATmega328 microprocessor (figure 3). Its working and usability is very similar in nature to the popular Arduino UNO – although it is smaller in size and is short of two Analog input/output pins. The device can be powered up by using the mini USB Jack, Vin Pin or +5V Pin. The mini USB jack can be connected to a phone/laptop charger which will supply +5V to the device (it is important to use a charger with voltage rating of +5V. If a charger with higher rating is used it is possible to burn out the device.) The Vin Pin is used for unregulated voltage in range 7-12V. The device takes in the input voltage and regulates it at +5V to run. The simplest option is to use a regulated

+5V power supply (example: batteries) and connect it to the board. For my project, I am using a regulated 12V power supply because it can power the microcontroller along with the RGB LED strip.

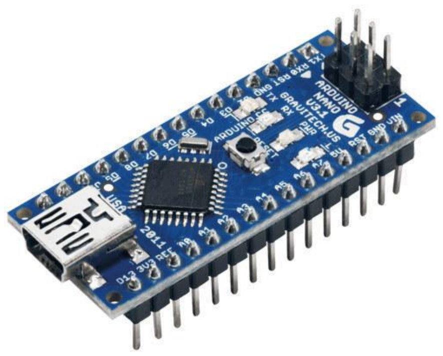


Figure 3: Image showing an Arduino Nano board
Source: Copied from [3]

The device has in total 36 pins (figure 4).

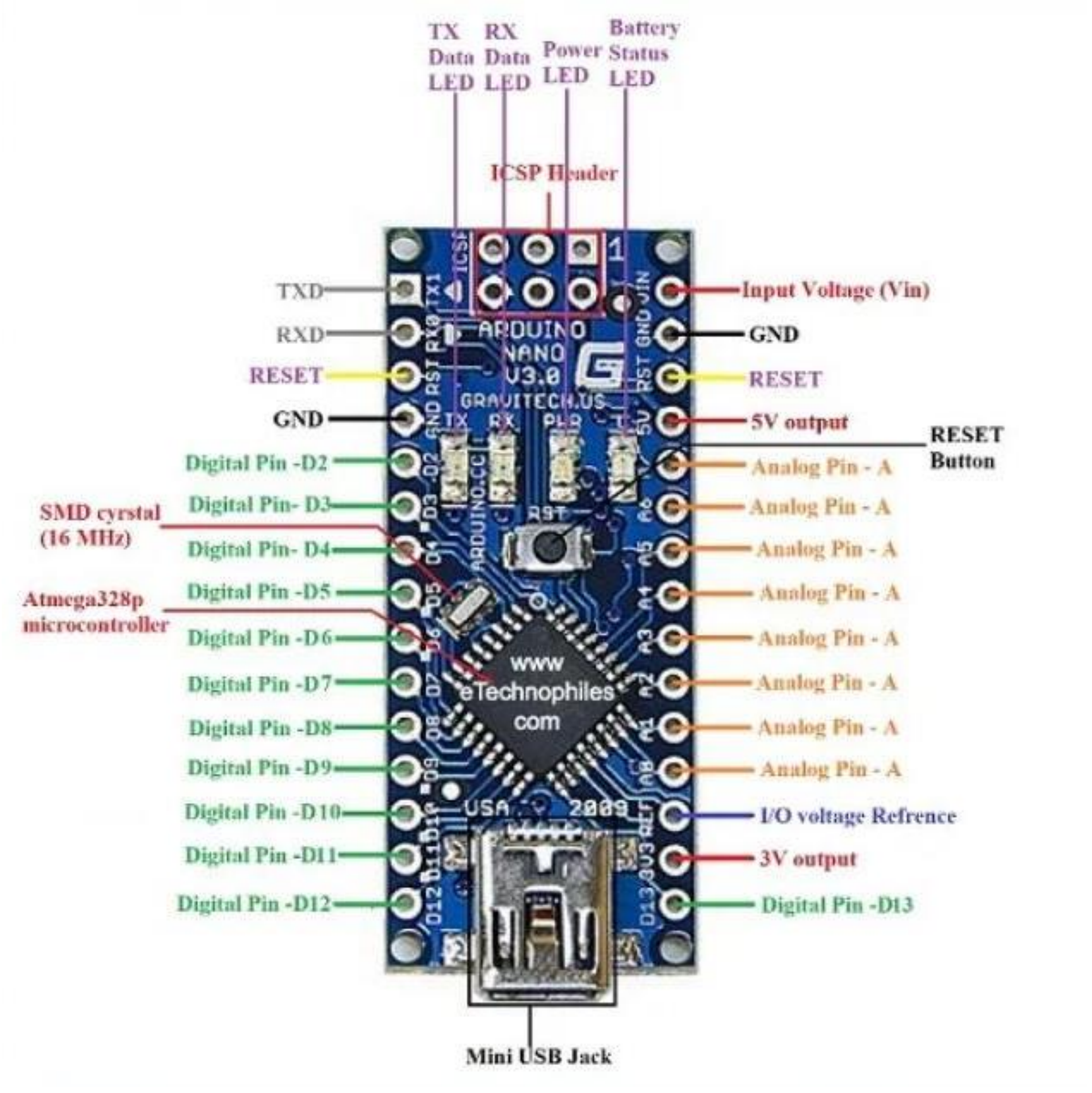


Figure 4: Image showing Arduino Nano Pin and Pout structure.
Source: Copied from [4]

The RXD and TXD pins are very important for our project purposes because they will used for communicating serially with the HC-05 module. The device has 8 Analog Input pins and fourteen digital Input/Output pins. Pins 3, 5, 6, 9, 10, and 11 can also be used as Pulse Width Modulation (PWM). They too are very important for our project because they allow us to generate an analog DC value using digital means. A digital output comprises of 0 volts (off) and 5 volts on Nano (on). The output voltage can be regulated by the pulse width, which is the on time of the digital output. A pulse width of 0 and 255 corresponds to voltage levels of 0V and 5V respectively (figure 5).

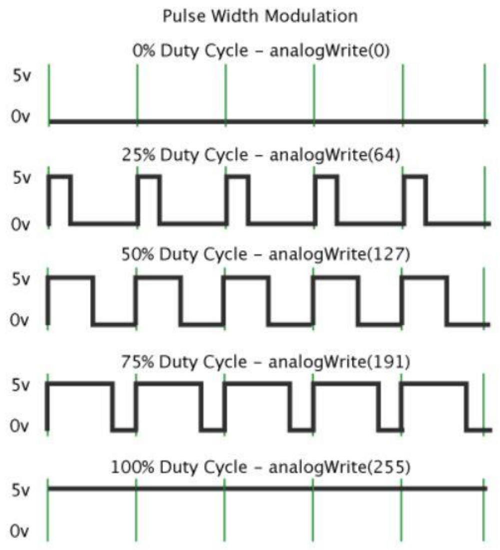

Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
25% Duty Cycle – analogWrite(64)
50% Duty Cycle – analogWrite(127)
75% Duty Cycle – analogWrite(191)
100% Duty Cycle – analogWrite(255)
5v
0v

Figure 5: Image showing pulse width modulation. As the pulse width shown in analogwrite('pulsewidth') increases, the output voltage increases.
Source: Copied from [5]

If the digital output changes very quickly then the voltage observed at the PMW pins is a steady voltage in range 0V to 5V.

Additionally, there is an RST pin on the board which is used to reset the device whenever 5V voltage is supplied to it. Moreover, the device can supply 3.3V and 5V using the 3.3V and 5V output pins. Arduino Nano also has four indicator LEDs [4]:

1. White LED: When ON, signifies that the board is sending data to the computer.
2. Red LED: When ON, signifies Arduino Nano is receiving data from the computer.
3. Blue LED: When ON, signifies high voltage supplied across digital pin 13 and vice-versa.
4. Power indicator: Signifies the status of battery.

The program is coded using an Arduino sketch in a C language. Since I had experience coding in Assembly language and Python, I was quickly able to learn sufficient C language to conduct this term project.

RGB LED strip (SMD 5050)

The device operates at 12V DC and is dimmable by applying appropriate voltages at the RGB channels. For my project, I have chosen the simple indoor SMD 5050 because it is more economical. Other options for the RGB strips included purchasing a waterproof led strip which can be installed outdoors and strips which had more LED density per unit meter. Since both incurred extra-unnecessary cost for my project, I purchased the SMD 5050. When purchasing I had to make another choice about the length. As the length of the strip increases, the number of components installed in the strip increase and, naturally, so does the price. This also meant the strip required more power to run as it must run more LEDs. Therefore, I opted to purchase a 1m length RGB LED strip. Figure 6 shows the SMD 5050.

Figure 6: Image showing SMD 5050.
Source: Copied from [6]

The internal workings of SMD 5050 can be modelled by figure 7.

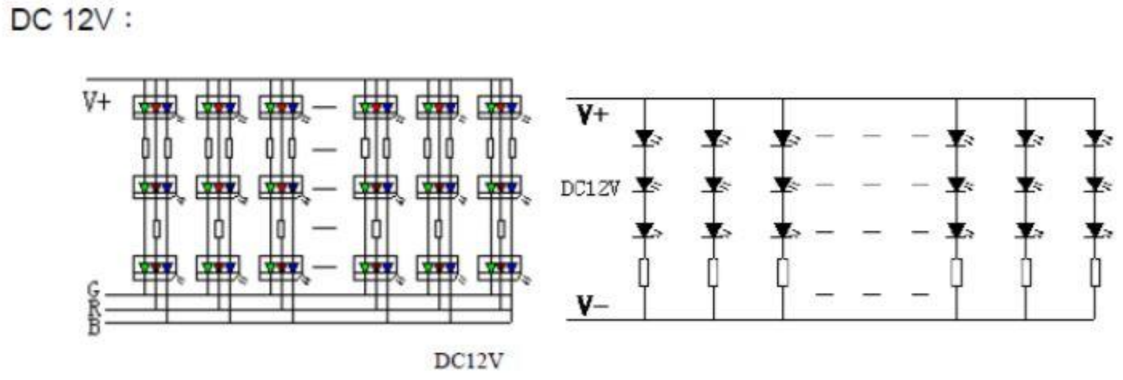


Figure 7: Image showing circuit diagram of SMD 5050.
Source: Copied from [7]

It can be seen from figure 7 that the strip has parallel and series connections. The device is made up of several short segments which comprise of three RGB LEDs. The RGB leds can be represented by a photodiode (shown on right in figure 7). If required, the wire can be cut into smaller pieces by cutting the wire after the end of a short segment. The circuit will still operate because we have simply removed some parallel connections. To increase the brightness of the strip, lower voltage should be supplied at the RGB channels as this will increase the potential difference across the photodiode and more current will flow.

NMOS Transistor

Figure 8: Image showing NMOS transistor
Source: Copied from [2]

The Metal Oxide Semiconductor Field Effect Transistor (MOSFET) conducts current by allowing electrons to flow through 'the channel' which is a conductive region generated due to voltage application at the gate terminal. As the voltage applied increases, the electric field generated increases which, in turn, attracts more electrons on the conductive path – 'the channel'. This relationship between electric field and conductive path also explain what the 'Field Effect' signifies in MOSFET. If the gate voltage supplied is positive, electrons gather from the p-type substrate which are in abundance of free electrons under the gate region and between the drain and source n-type electrodes, allowing current to flow from drain to source. When such behavior is observed we have an n-channel MOSFET (NMOS). As is apparent from the previous discussion, increasing gate voltage will attract more electrons and make the conductive channel wider, thus increasing drain current (because it flows from drain to source) and vice-versa. Once a suitable level of gate voltage supplied and conductive channel constructed, the transistor is ON and allows drain current to flow. Because we are concerned with inverting the input signal for our project purposes, we will now acquaint ourselves with using MOSFETs as a switch.

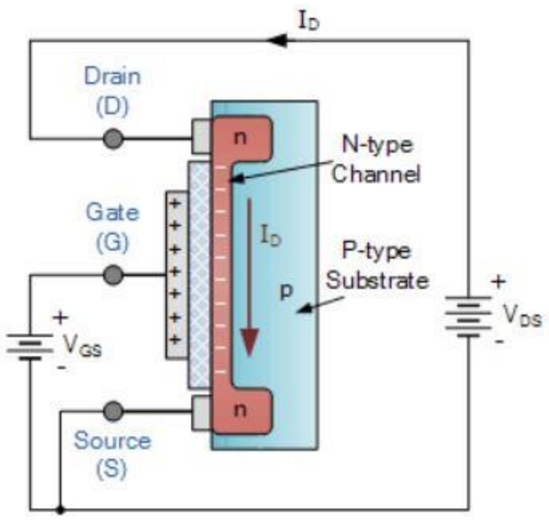


Figure 9: Image showing how an NMOS operates: the gate voltage is supplied with positive voltage which then attract electrons from the p-substrate, forming the conductive channel. This allows current to flow from drain to source

The NMOS (shown in figure 10) is OFF when Vi < Vth. The output voltage or the voltage supplied at the RGB channel will be same as Vs because no current flows and thus no potential energy lost. When Vi > Vth current flows from drain to source. When the NMOS operates in active region, the current is modelled by:

	$\mathrm{I_D} = K_N(V_{GS} - V_{TN})^2$	(1)

	$\mathrm{I_D} = K_N(V_{input} - V_{TN})^2$	(2)

The NMOS stays in active region when:

$$V_{\mathrm{DS}}(sat) = V_{GS} - V_{TN} \quad (3)$$

Equation 3 can further be simplified down to:

$$V_{\mathrm{D}}(sat) = V_{G} - V_{TN} \quad (4)$$

Therefore, when $V_{\mathrm{DS}} > V_{\mathrm{DS}}\ (sat)$, the transistor is in active region.

$$V_{\mathrm{D}} = V_{\mathrm{S}} - \mathrm{I_D} \, . \, \mathrm{R_D} \quad (5)$$

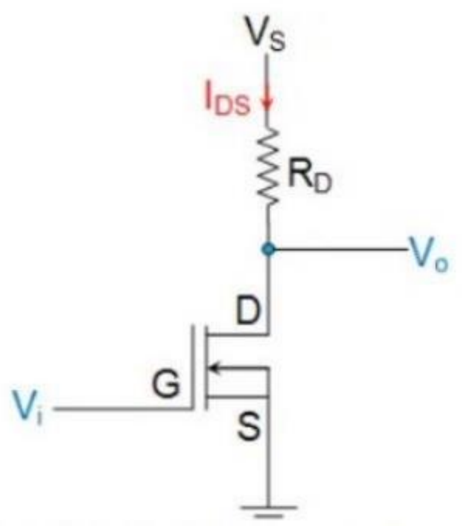


Figure 10: Image showing N-channel MOSFET as a switch.
Source: Copied from course book

Arduino Output

An Arduino is only capable of producing only binary voltages (high and low). The specific Arduino model that is being used in the project can produce at maximum 4.72V and minimum 0V. Since we will be using a mobile application to control the colors of the LED, we must use the Arduino as it allows us to communicate with the mobile application using Bluetooth module. To change the colors on the LED we must somehow supply different amounts of voltages across the LED strip. This is not possible as previously indicated that the Arduino is possible of producing only binary voltages. However, by changing the binary voltages fast enough, we can change the brightness of the LED strip.

The time during which the signal is high is called "On time". Similarly, the time during which the signal is low is called "Off time". The sum of On and Off time is the total period of the signal. The duty cycle is the ratio of On time with total period of the signal. Figure 1 demonstrates the notion.

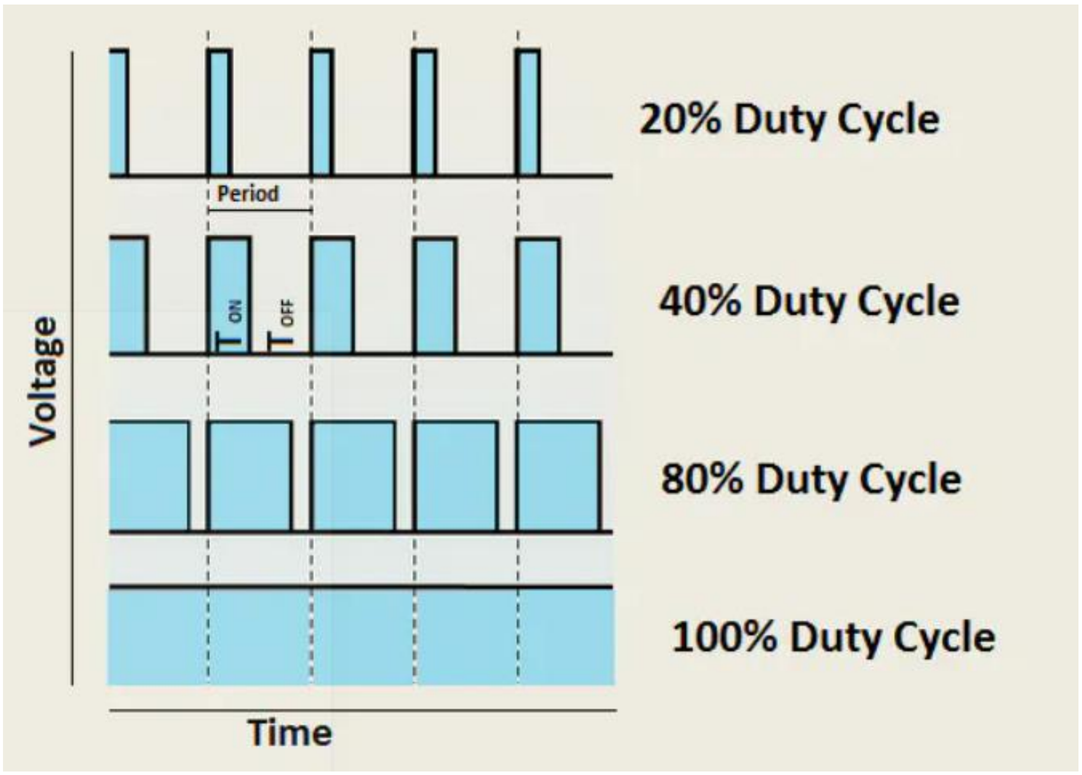


Figure 11: Image showing different duty cycles and the behavior of the signal.
Source: Arduino

The brightness of the LED can be changed using this duty cycle concept. For example, at 20% Duty cycle and 1Hz frequency, the LED is on for 0.2 seconds and off for 0.8 seconds. Similarly, at 50% Duty cycle and 10Hz frequency, the LED is on 10 times and off 10 times during a second. If the frequency is fast enough, the LED turns on and off faster. Since the human eye is unable to distinguish the LED turning on and off, it appears to us that the brightness of the LED has reduced.

Therefore, using the pulse width modulation we can change the brightness of the RGB LED strip. The problem is that this causes the MOSFETs which are between the Arduino and the RGB LED strip to turn on and off several times in a second, depending on the frequency. For example, at 50% duty cycle and 10 Hz frequency, the MOSFET turns on and off 10 times each. This is the because the V_{TN} of the MOSFETs we are using in the project (IRFZ44N) is around 0.06V. Therefore, when V_{GS} = 0V – where source voltage is 0V – the transistor turns off. This behavior is undesirable as the project specifications state that we need at least one MOSFET in saturation region. To accommodate this project specification, we will first look at the schematic of our project and then delve into the solution of this problem.

Figure 12: Image showing schematic of the Bluetooth Controlled RGB LED Strip

As figure 2 shows, we are using 3 MOSFETs for the project. Each MOSFET is responsible for a channel (red, green, and blue). To ensure that our project is dynamic in nature and allows us to view different colors, we will keep **only 1** MOSFET in saturation region. This can be achieved by operating one of the Arduino output at 100% Duty cycle (On for the entire period). This way the MOSFET will never turn off. Next, we wish to decrease the voltage supplied at the gate of the MOSFET. This can be shown by the following working:

$$\mathrm{I_D}\ (Saturation\ current) = K_N(V_{GS} - V_{TN})^2, where\ Vs = 0V \tag{6}$$

$$\mathrm{I_D} = K_N(V_G - V_{TN})^2, where\ Vg = output\ of\ Arduino \tag{7}$$

The critical point for NMOS to stay in active region is:

$$V_{\mathrm{DS}}(sat) = V_{GS} - V_{TN} \tag{8}$$

Equation 3 can further be simplified down to:

$$V_{\mathrm{D}}(sat) = V_G - V_{TN}, where\ V_{TN} = 0.06V \tag{9}$$

Equation 4 is approximately.

$$V_{\mathrm{D}}(sat) = V_G \tag{10}$$

Therefore, when $V_{\mathrm{DS}} > V_{\mathrm{DS}}\ (sat)$ or $V_{\mathrm{D}} > V_{\mathrm{G}}$, the transistor is in active region.

Before we move on, we need to understand for what V_{G} , the V_{D} is greater. This can be understood by looking at the following experimental images:

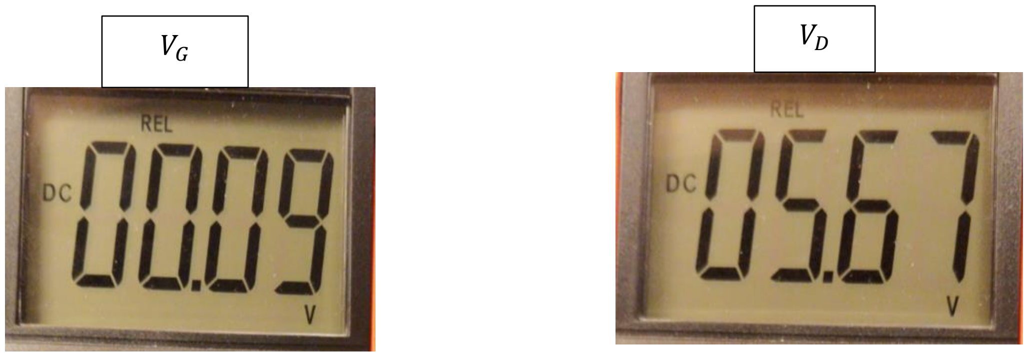


Figure 13: Image showing Gate and Drain voltage when pulse width is 5.

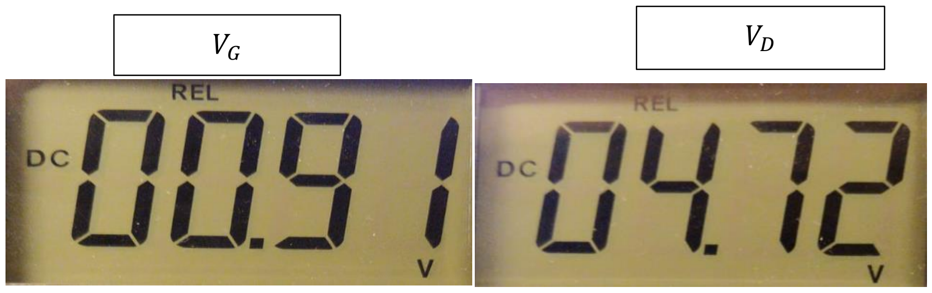


Figure 14: Image showing gate and drain voltage when pulse width is 50.

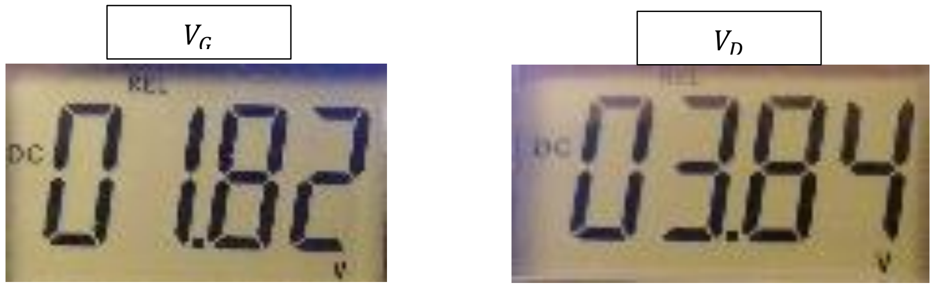


Figure 15: Image showing gate and drain voltage when pulse width is 100.

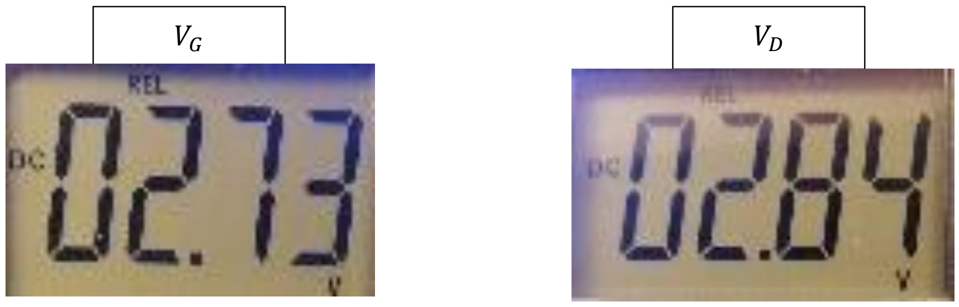


Figure 16: Image showing gate and drain voltage when pulse width is 150.

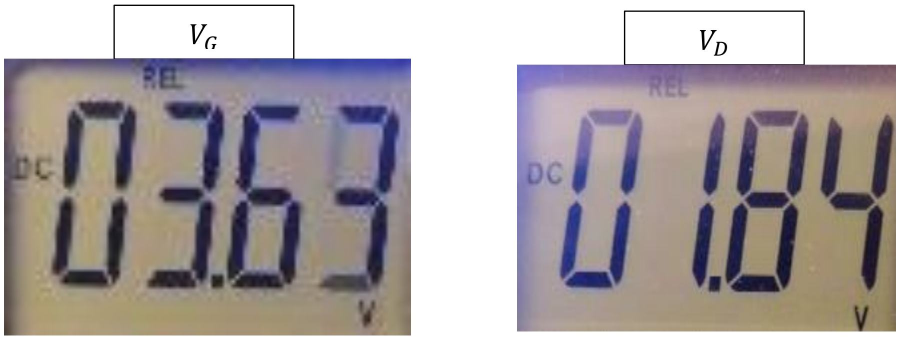


Figure 17: Image showing gate and drain voltage when pulse width is 200.

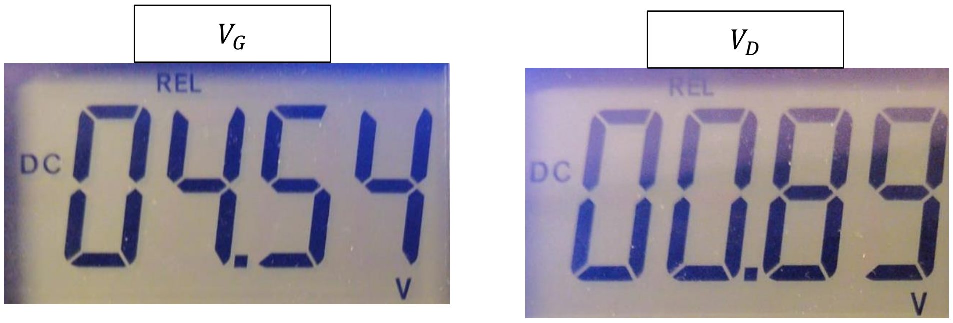


Figure 18: Image showing gate and drain voltage at an NMOS when pulse width is 250.

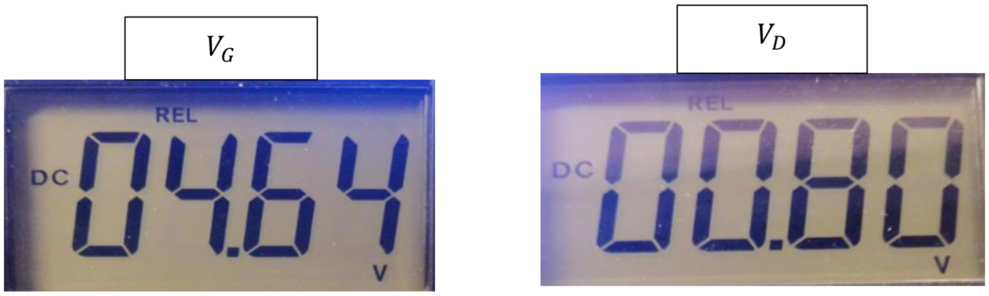


Figure 19: Image showing gate and drain voltage at an NMOS when pulse width is 255.

The results are from the multimeter readings are summarized below:

Table 1: The table shows voltage values of gain and drain of an NMOS measured against different pulse width using a multimeter. All the NMOS react to the pulse width similarly.

Pulse Width	V_G (V)	V_{DS} (V)	MOSFET Region
5	0.09	5.67	SAT
50	0.91	4.72	SAT
100	1.82	3.84	SAT
150	2.73	2.84	SAT
200	3.63	1.84	NON-SAT
250	4.54	0.89	NON-SAT
255	4.64	0.80	NON-SAT

It is explicitly shown in figure 6 and 7 that as Vg increases from 2.73V to 3.63V, the drain voltage decreases from 2.84V to 1.84V. Therefore, the MOSFET goes from saturation to non-saturation region from figure 6 to 7. Finally, we now know what voltage we need to supply to Vg of MOSFET to ensure that it is in active region. To achieve this desired voltage, we need to pass the square signal generated by the Arduino through a low pass RC filter.

The DC component of the square pulse generated by the Arduino is:

$$\text{DC component} = 4.72(\text{Duty Cycle}) \tag{11}$$

, where the duty cycle is:

$$\text{Duty Cycle} = \frac{T_{ON}}{T_{ON} + T_{OFF}} \tag{12}$$

For the purposes of our project, we are interested in a square pulse (highs and lows) and wish to extract the steady DC component. Therefore, we will now look at an example square pulse and observe its frequency components.

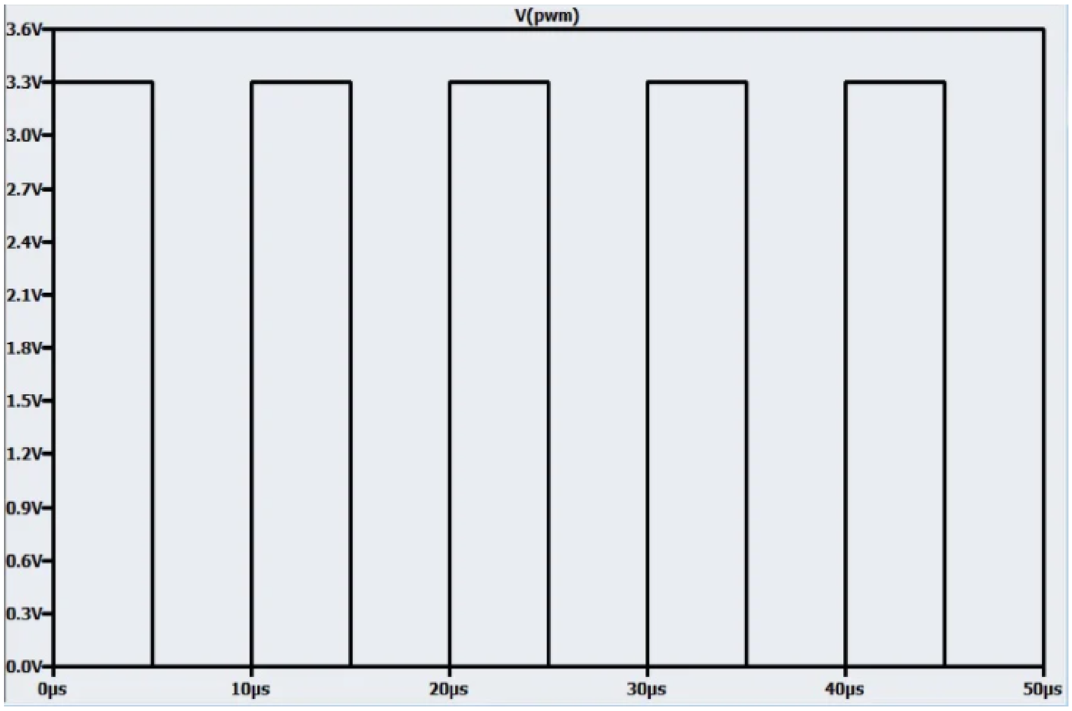


Figure 20: Image showing a square pulse.
Source: All about circuits

The square pulse under consideration (figure 20) has an amplitude of 3.3V and time period of 10µ seconds. The frequency components of this square pulse are shown in figure 12.

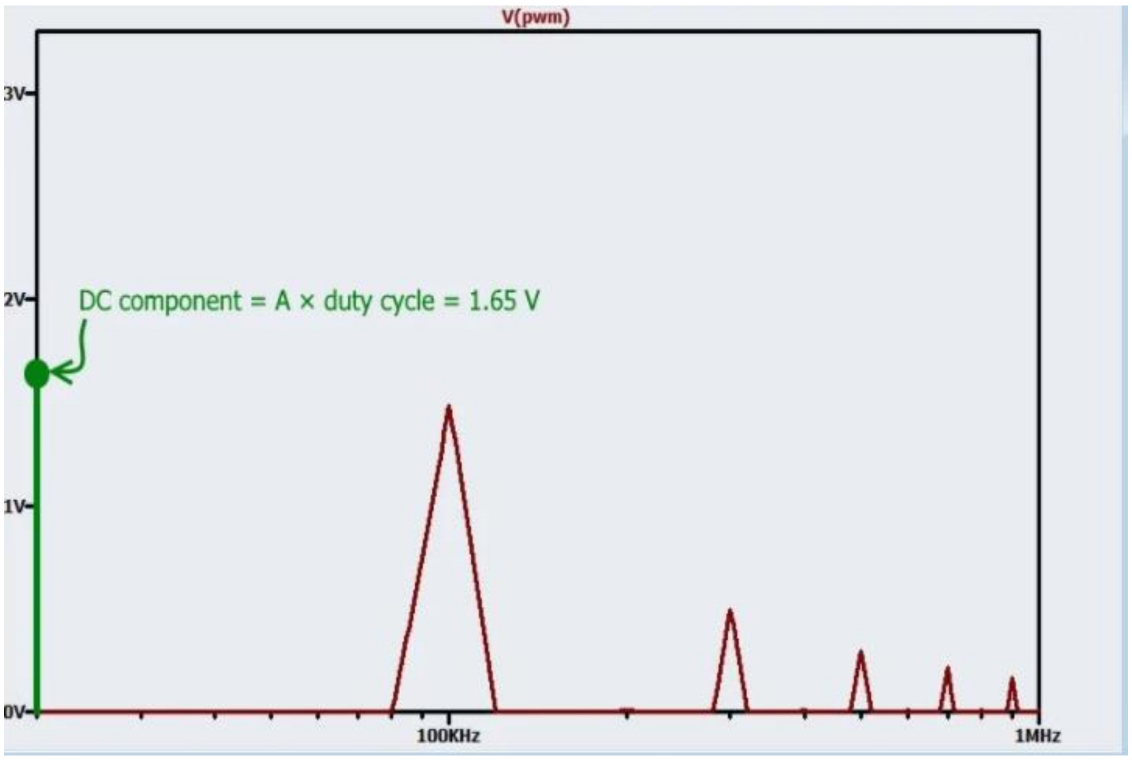


Figure 21: Image showing frequency components of a square pulse.

The DC component is found at 0Hz. We then observe frequency components at the odd multiples of carrier frequency e.g.: 100KHz, 300kHz, etc. Since we wish to operate our MOSFET in saturation region and want it to be stable, we extract only the DC part of the signal. Consequently, we use the RC low pass filter. Since it practically impossible to create a perfect filter which stops frequency components after the corner frequency, the filter signal will be more representative of the DC component if the carrier signal is high. This will ensure the undesired carrier components of the signal are attenuated more and, in turn, influence our filter signal less.

A low pass filter attenuates frequency components that are above the cut off frequency. Figure 10 shows the bode plot of a low pass filter. A bode plot describes how the gain varies with respect to the frequency.

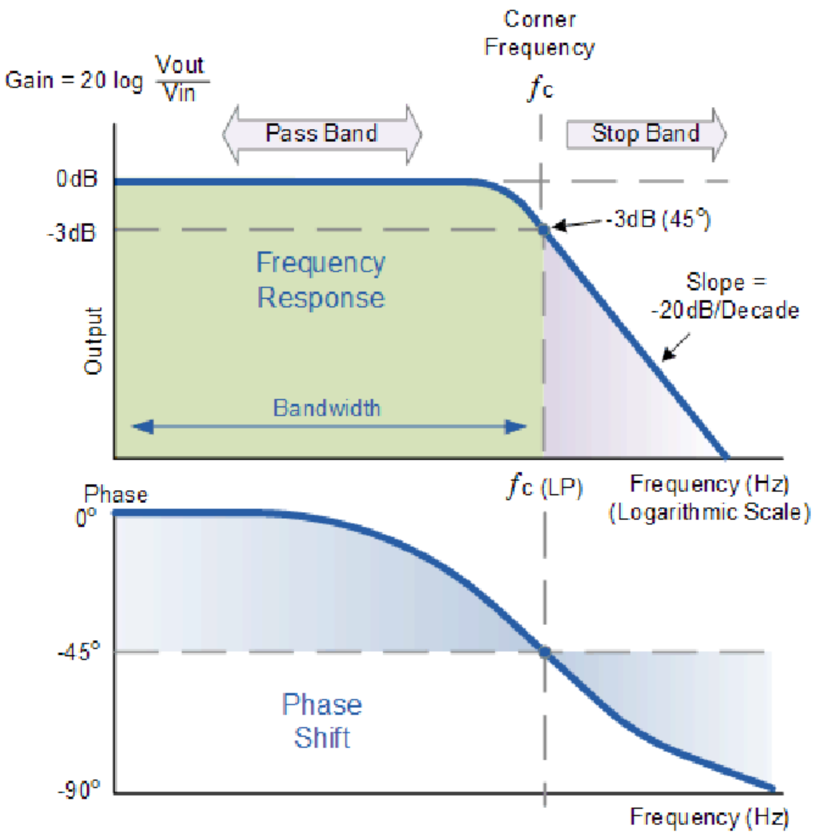


Figure 22: Image showing the bode plot of a low pass filter.
Source: Passive components blog

It can be seen on figure 12 that for low frequencies the signals are allowed to pass with minimal attenuation. However, after the corner/cutoff frequency, the signal experiences more and more gain. An RC circuit is a low pass filter which will be deployed in this project to filter out the DC component.

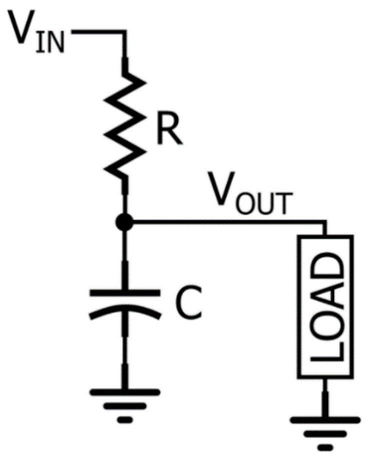


Figure 23: Image showing an RC circuit.

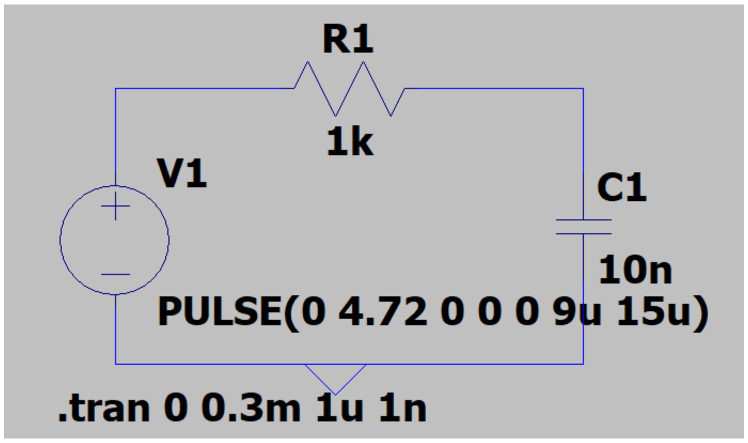
When the frequency is low, the reactance of the capacitance increases causing more voltage to drop across it and the load. Contrarily, when the frequency of input signal is high, the reactance of the capacitance decreases. Consequently, more voltage is dropped across the resistor. In other words, low frequencies are allowed to pass while high frequencies are stopped. Since we wish to extract the stable DC component of the square pulse, our corner frequency should be close to 0. Additionally, we want the carrier frequency to be as high as possible to ensure the carrier components experience more attenuation.

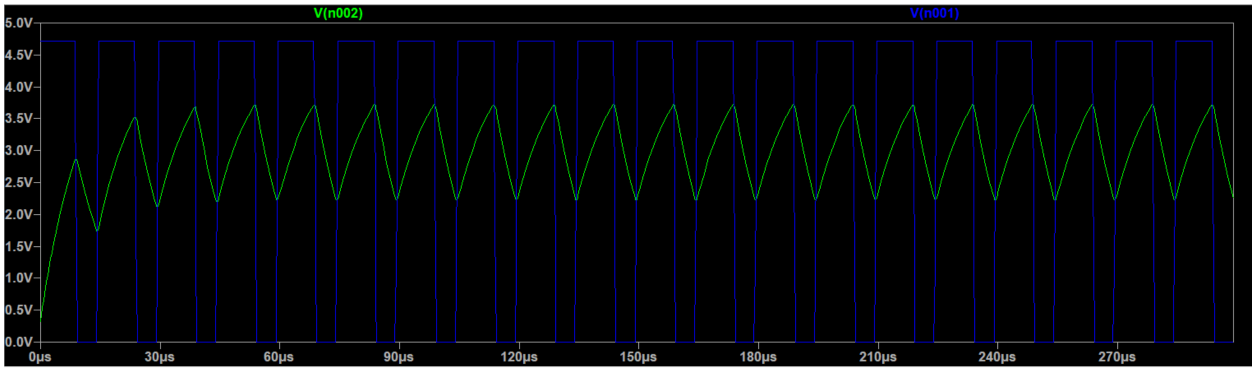
The cut off frequency of an RC circuit is defined by the following equation:

$$f_c = \frac{1}{2\pi RC} \tag{13}$$

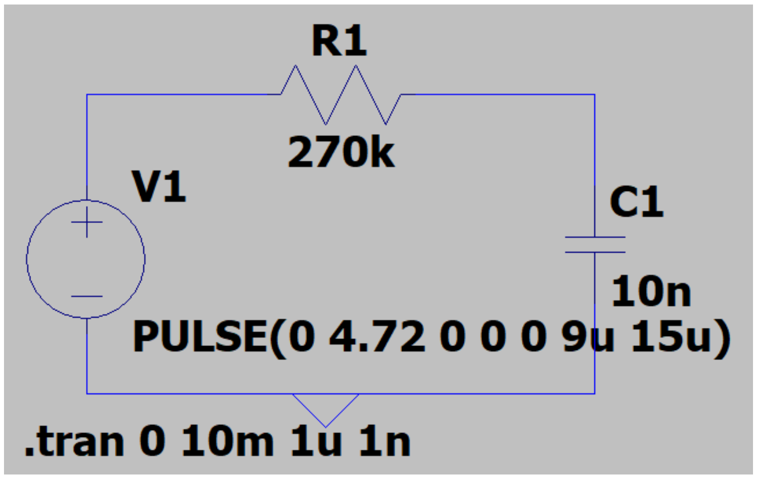
To investigate our circuit behavior, we will use a capacitor of 10n and change the resistor values to accommodate different cut-off frequencies. This is because resistors are available in more range of values than capacitors. It is important to highlight it again that we wish to only retrieve the DC component of our signal which is found at the origin of the frequency domain plot, where frequency is 0. The cut-off frequency closer to 0 and the carrier frequency further away from 0, the better filtering of the DC component. Now, we will investigate the behavior of the low-pass filter by simulating it on LTSpice.

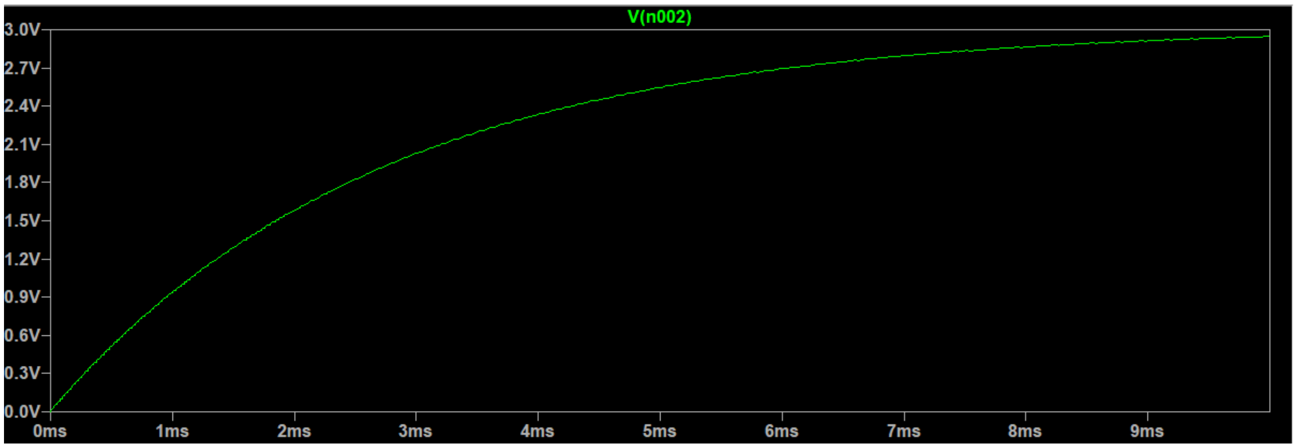
Firstly, we will use a low resistor value to observe how it affects the output as this will drive the cut off frequency to be high.





The output of the low-pass filter is very unstable because the cut-off frequency is 15k Hz. This is not a stable DC output that we want. Therefore, we will increase the resistor value to make the circuit's output more stable.





As the R values increases, the cut off frequency decreases. Therefore, the AC components of the square pulse are attenuated more, and we get a more stable output from the low-pass filter. Now, we will implement this low-pass filter in hardware and consider the noise as well into our calculations before deciding on the values of desired low-pass filter.

If the cut off frequency is 15 kHz, then we will use a resistor of value 1 kΩ. The results obtained are the following:

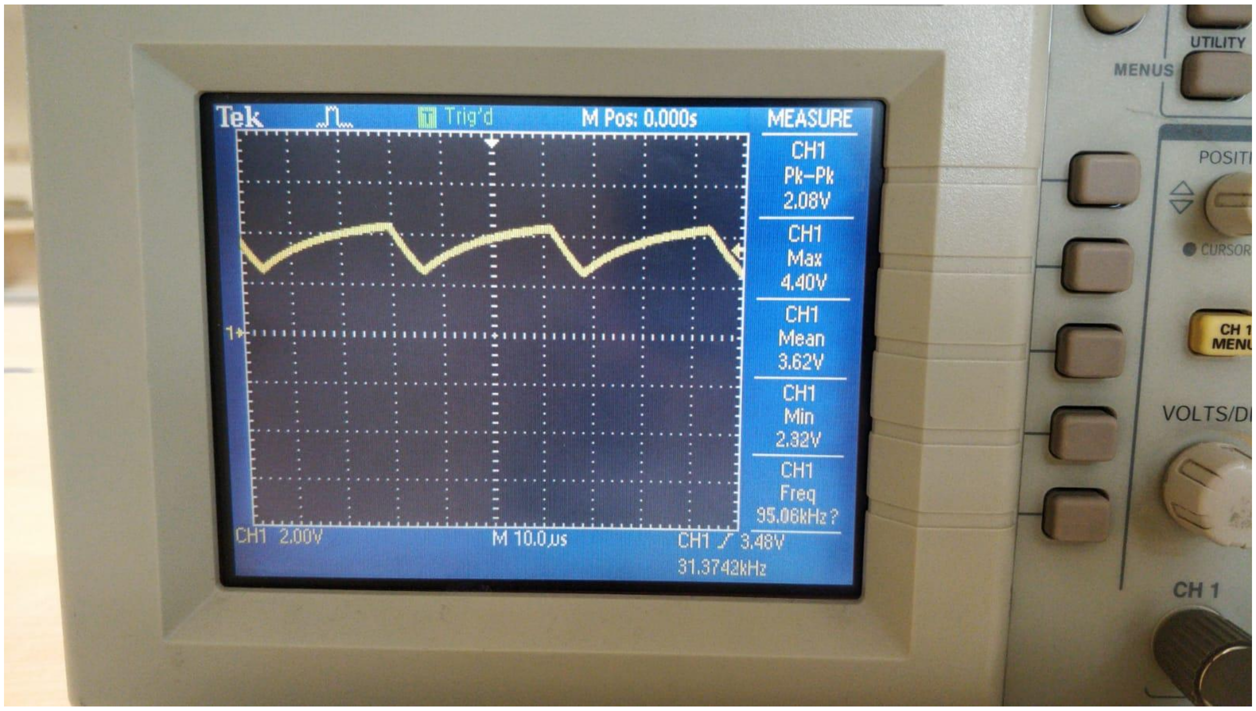


Figure 24: Image showing the DC output of the low-pass filter with R = 1k and C = 10n.

The DC value is relatively much more stable than

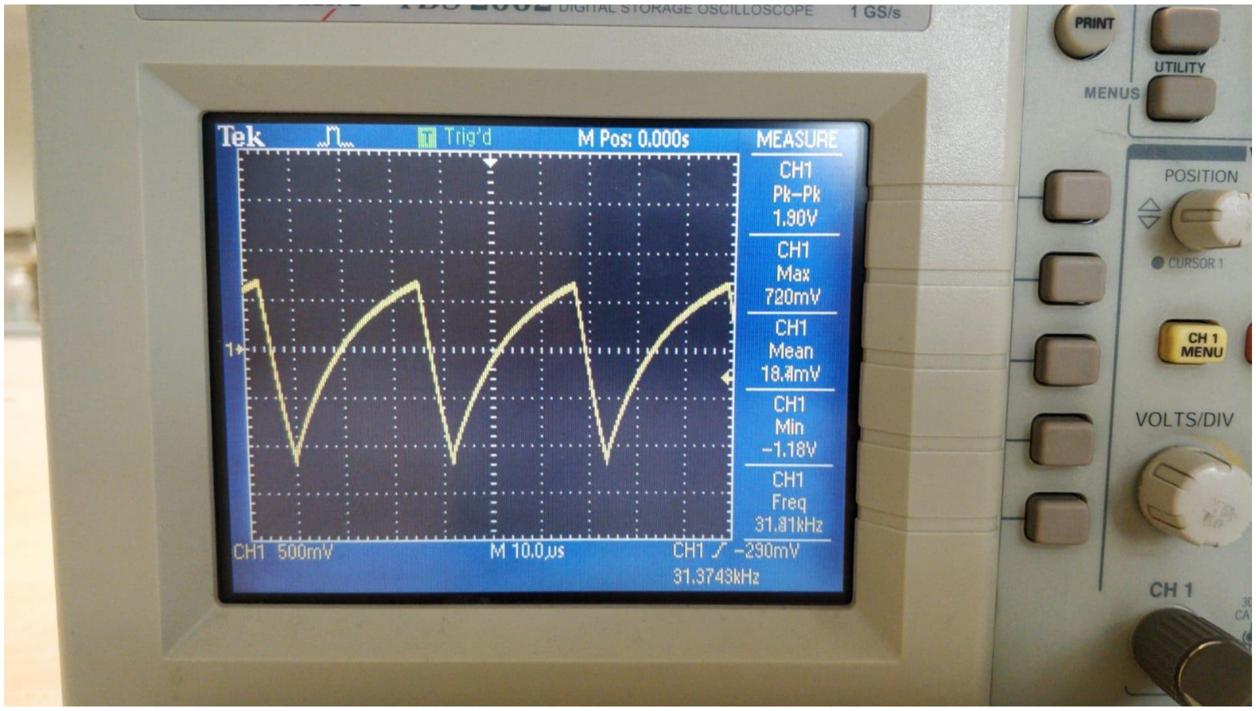


Figure 25: Image showing the AC output of the low-pass filter with R = 1k and C = 10n.

If the cut-off frequency is 1 kHz, then we will use a resistor value of 15 kΩ and capacitor value will remain 10n. The results obtained are shown below:

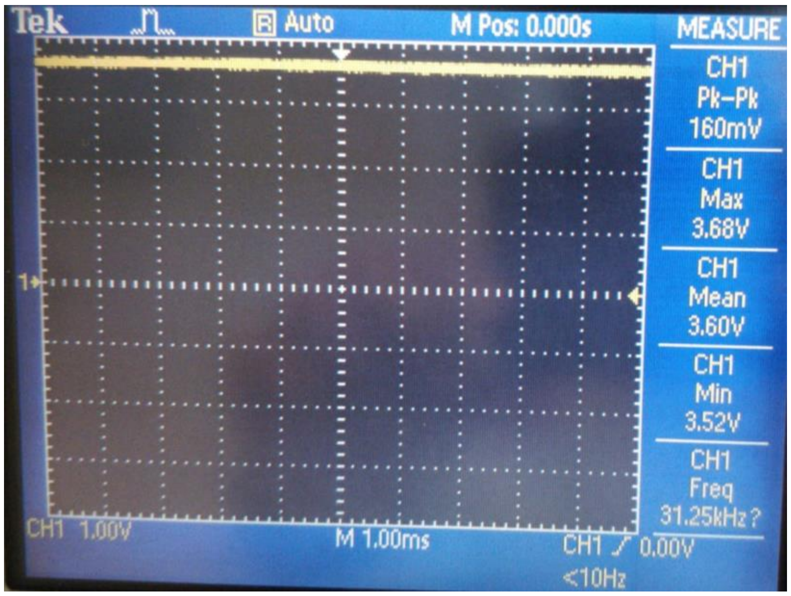


Figure 26: Image showing the DC output of the low-pass filter when the cut-off frequency is 1 kHz.

The DC value is much more stable than the one obtained with a low-pass filter with cut-off frequency 15 kHz. This observation agrees with our hypothesis of cut-off frequencies closer to 0Hz filtering the DC component better and attenuating the undesired frequency components of our input signal more. The AC output of the signal also aligns with the theory.

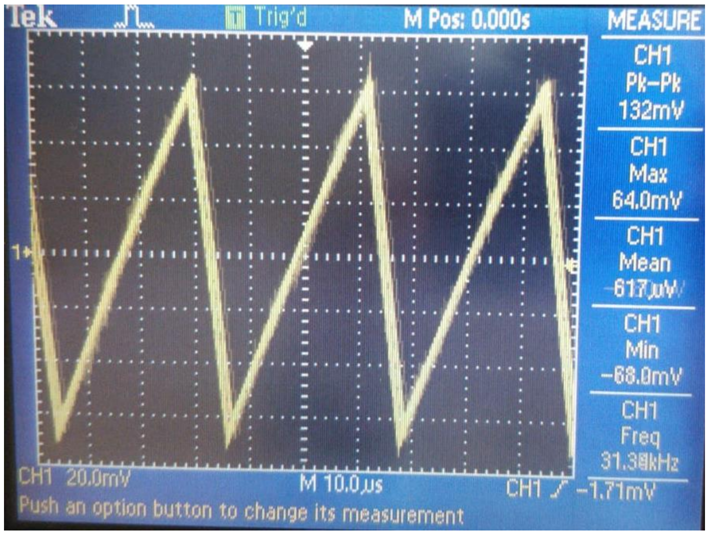


Figure 27: Image showing the AC output of the low-pass filter with cut-off frequency of 1 kHz.

It can be explicitly seen from the above picture that the AC signal variations are much less in comparison to the AC signal output of a low pass filter with cut-off frequency 15 kHz. The Pk-Pk voltage for the AC signal components for this low-pass filter is 132 mV which is 14x less than the AC signal output of a low pass filter with frequency 15 kHz. This observation implies that AC components of the signal are attenuated more by this filter, indicating that the low-pass filter with cut-off frequency of 1 kHz is much superior to one with cut-off 15 kHz.

Next, we will try to make the output DC even more stable and attenuate the AC more by introducing a resistor of much higher value e.g.: 270 kΩ. The cut-off frequency for this low-pass filter is 50 Hz. The following images show the AC and DC outputs of the low pass filter.

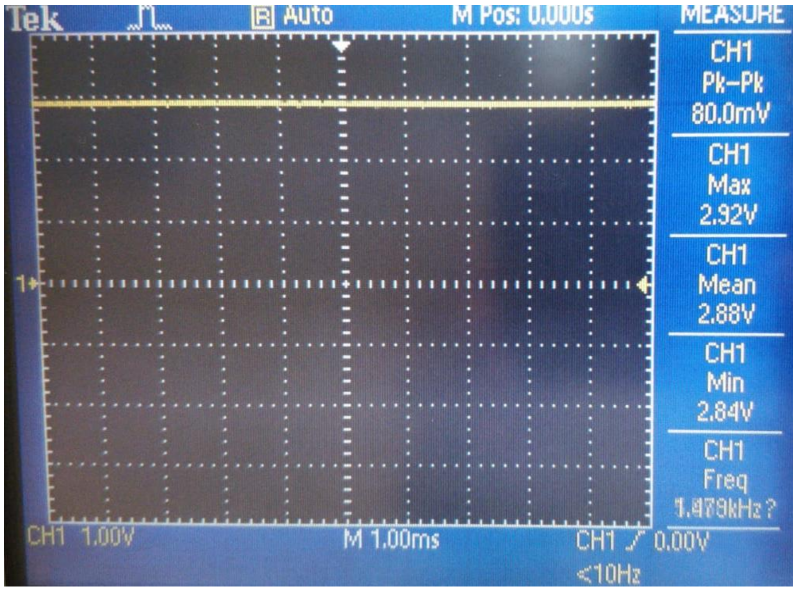


Figure 28: Image showing the DC output of a low-pass filter with cut-off frequency 50 Hz.

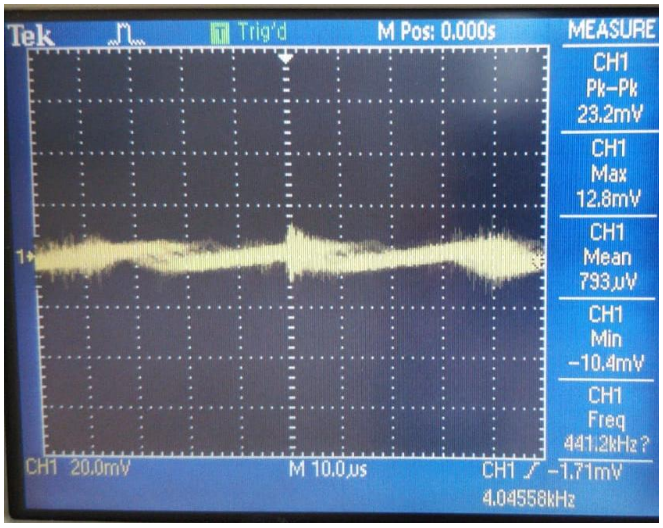
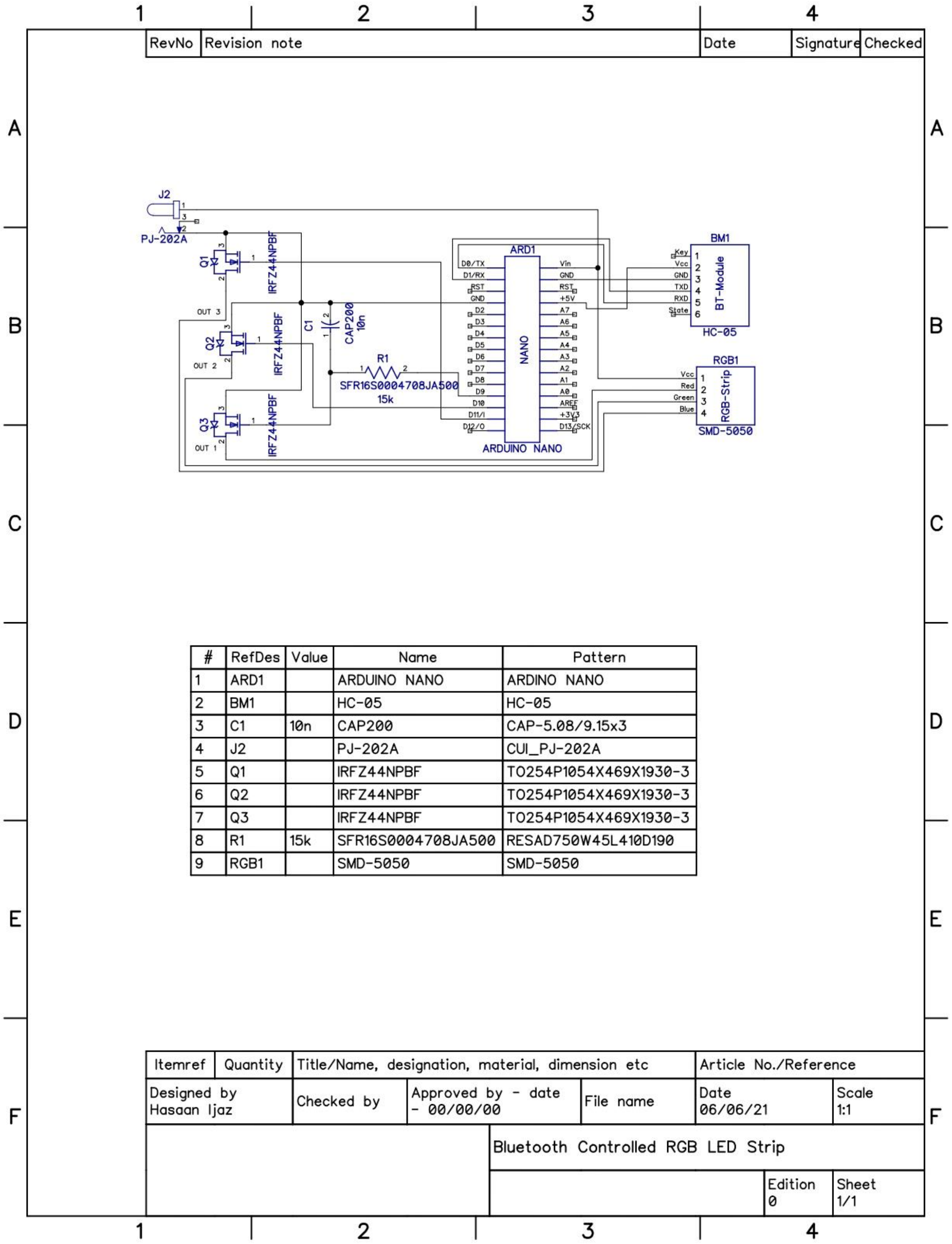


Figure 29: Image showing the AC output of a low-pass filter with cut-off frequency 50 Hz.

Therefore, for our project purposes we chose a low pass filter whose resistor is 15 kΩ and capacitor is 10nF as it allows us to extract the stable DC component of the pulse width modulation. It is important to highlight that the oscillations in the output are negligible and can not be seen on the RGB LED Strip. For a low pass filter with resistor value of 1kΩ and capacitor 10nF , the output voltage oscillates between large values. This causes flickering on the RGB LED strip.

Schematic

Finally, all the components can be brought together and connected as shown in figure 11 to achieve the desired functionalities.



#	RefDes	Value	Name	Pattern
1	ARD1		ARDUINO NANO	ARDINO NANO
2	BM1		HC-05	HC-05
3	C1	10n	CAP200	CAP-5.08/9.15x3
4	J2		PJ-202A	CUI_PJ-202A
5	Q1		IRFZ44NPBF	TO254P1054X469X1930-3
6	Q2		IRFZ44NPBF	TO254P1054X469X1930-3
7	Q3		IRFZ44NPBF	TO254P1054X469X1930-3
8	R1	15k	SFR16S0004708JA500	RESAD750W45L410D190
9	RGB1		SMD-5050	SMD-5050

Itemref	Quantity	Title/Name, designation, material, dimension etc		Article No./Reference	
Designed by Hasaan Ijaz	Checked by	Approved by - date - 00/00/00	File name	Date 06/06/21	Scale 1:1
			Bluetooth Controlled RGB LED Strip		
				Edition 0	Sheet 1/1

Figure 30: Image showing the schematic of the proposed circuit.

Hardware

Firstly, the circuit proposed is assembled on the breadboard without the Bluetooth module. The purpose of this circuit was to confirm that the microcontroller is communicating with the RGB LED strip. The hardware setup is as shown below:

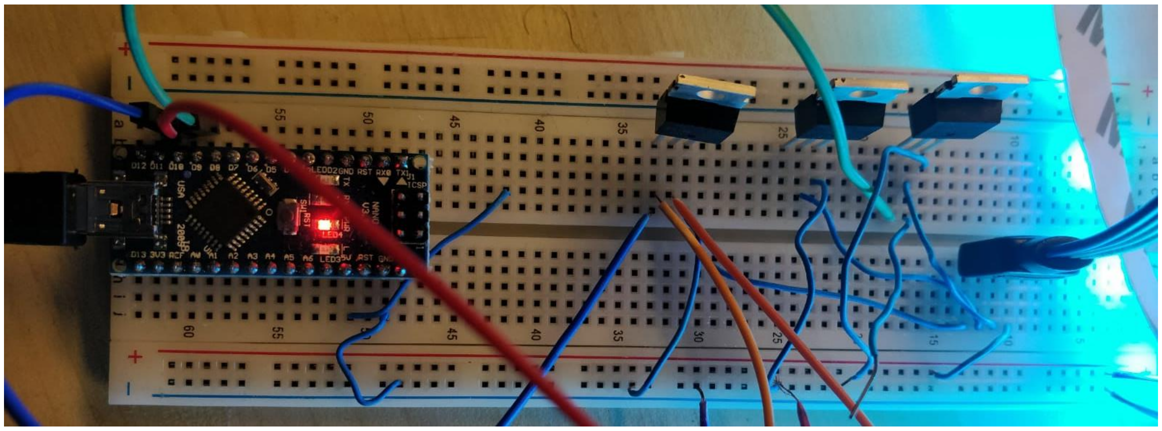


Figure 31: Image showing hardware implementation of the circuit proposed.

One the components have been installed on the breadboard; I test the RGB LED strip by observing the brightness of the LED strip when only one channel of the RGB strip is high.

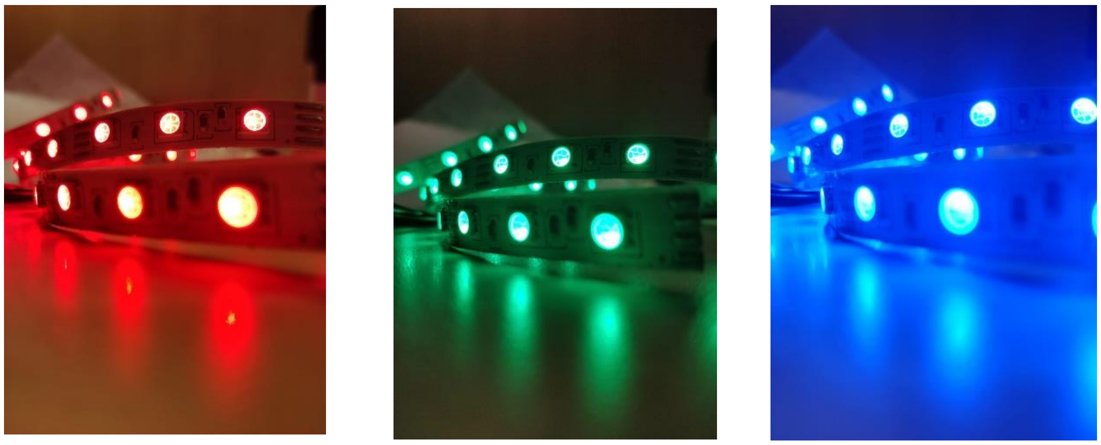


Figure 32: Image showing the RGB strip when voltage is applied across only one channel. The left image is observed when the pulse width is 255 and forwarded to red NMOS while the pulse width for the other two NMOS or Green-Blue channels is 0. Similarly, the other two colors are observed by setting their corresponding pulse width to 255 (generates highest possible voltage level) and giving it to their corresponding NMOS transistors and setting the remaining channels to 0.

Once I had determined the hardware implementation was working, I decided to code 'Fading LEDs.' This was done in Arduino Sketch and the code written (Appendix.1) supplies varying pulse width at the three channels such that we observe a fading effect; in other words, we observe different colors of the RGB spectrum continuously. This effect has been recorded and put up on my Youtube channel. It can be accessed by clicking on **demonstration**.

Next, I incorporated the Bluetooth module and the low-pass filter in my breadboard implementation of the circuit as well. The following images show the new circuit:

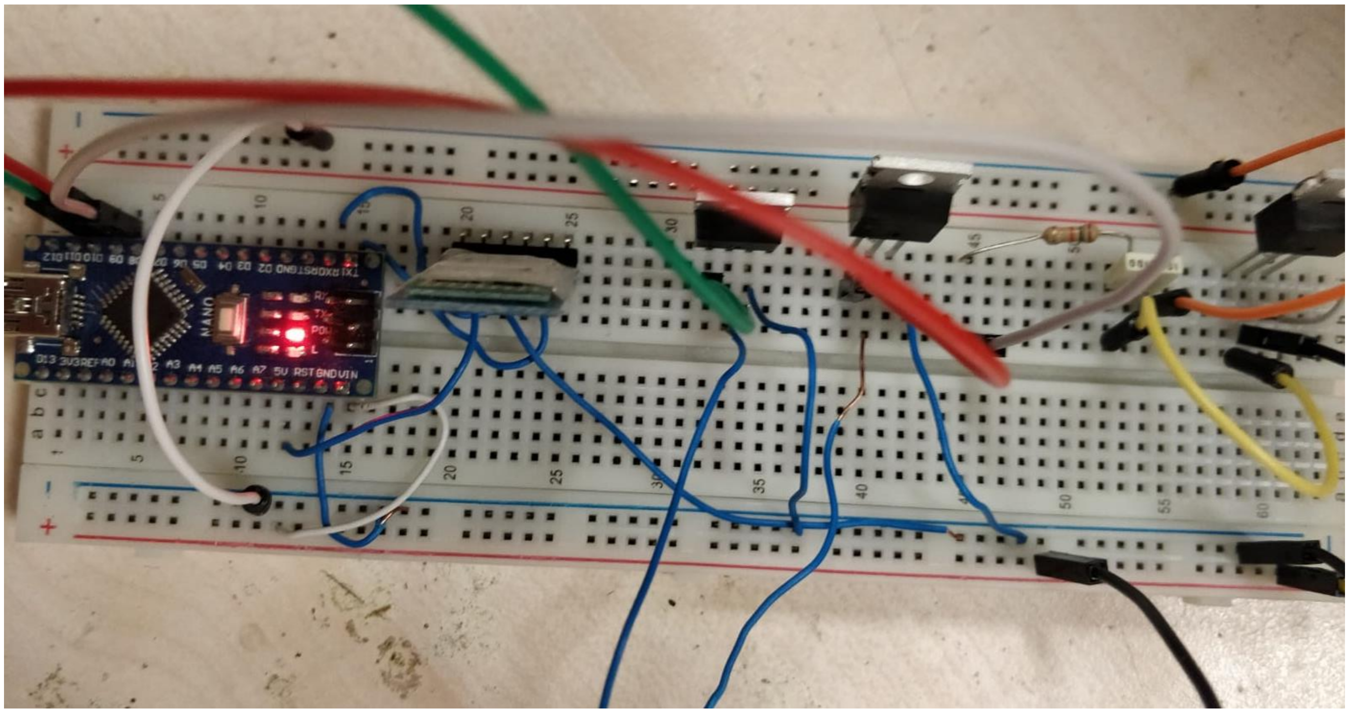
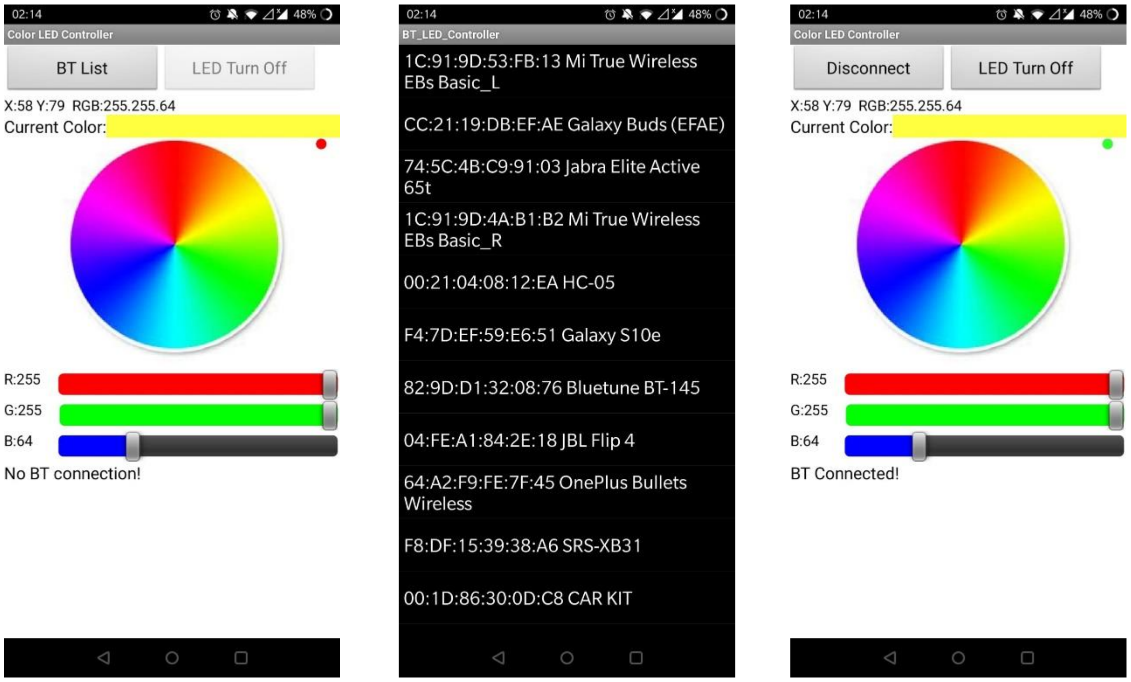


Figure 33: Image showing the breadboard implementation of proposed circuit.

To communicate with the Bluetooth module, a mobile application designed for RGB LED strip was installed from the play store and exploited [9].



X:58 Y:79 RGB:255.255.64
Current Color:
R:255
G:255
B:64
No BT connection!

1C:91:9D:53:FB:13 Mi True Wireless EBs Basic_L
CC:21:19:DB:EF:AE Galaxy Buds (EFAE)
74:5C:4B:C9:91:03 Jabra Elite Active 65t
1C:91:9D:4A:B1:B2 Mi True Wireless EBs Basic_R
00:21:04:08:12:EA HC-05
F4:7D:EF:59:E6:51 Galaxy S10e
82:9D:D1:32:08:76 Bluetune BT-145
04:FE:A1:84:2E:18 JBL Flip 4
64:A2:F9:FE:7F:45 OnePlus Bullets Wireless
F8:DF:15:39:38:A6 SRS-XB31
00:1D:86:30:0D:C8 CAR KIT

X:58 Y:79 RGB:255.255.64
Current Color:
R:255
G:255
B:64
BT Connected!

Figure 34: Image showing the Bluetooth mobile application used to control the RGB LED strip. The leftmost image shows the unpaired mobile application. The center image shows the options displayed on the BT List button. The rightmost image shows the mobile application paired with HC-05.

To communicate with the Bluetooth module and this specific mobile application, an Arduino code was found from the designer’s website and adapted to our needs (see Appendix. 2).

As expected, one of the MOSFETs always stayed in the saturation region. This was achieved by constraining the pulse width of the red channel to 100.

Table 2: Table showing the pulse width across the red channel of the RGB LED strip and the gate voltage observed at the corresponding MOSFET.

Pulse width red	V_R (V)
99	2.07
255	2.09

Table 3: Table showing the pulse width across the green channel of the RGB LED strip and the gate voltage observed at the corresponding MOSFET.

Pulse width green	V_R (V)
99	2.07
255	5.13

Table 4: Table showing the pulse width across the blue channel of the RGB LED strip and the gate voltage observed at the corresponding MOSFET.

Pulse width blue	V_R (V)
64	1.382
255	5.13

As expected despite increasing the pulse width of red channel to 255, the red channel only shows 2.09V because when a pulse width higher than the constrained one is observed it is replaced by the maximum of constrained pulse width which in our case is 100. Contrarily, when the pulse width is 255 at the other channels, the gate voltage observed is 5.13V. For pulse width within the constraints, the voltage decreases for all MOSFETs.

It is important to highlight that the low-pass filter has been implemented in this circuit so the MOSFET responsible for the red channel is always in SATURATION region. By controlling the gate voltage using Appendix. 2, we ensure that the drain voltage of the red channel is always higher than the gate voltage. After concluding all the components work together in harmony, we move on to design the PCB using DipTrace.

When designing the DipTrace special attention was placed to the placement of components on the PCB. This was done to ensure no jumper wires were used and the traces were well spaced from each other. Additionally, new patterns that were not found in the DipTrace library like the Arduino, power jack, Bluetooth module and RGB LED strip were either constructed manually or their corresponding sketches were found online. To ensure that the circuit could be represented in a black box implementation, dimensions of the PCB were strictly monitored to match those of the box we were instructed to use. The mounting holes on the PCB were also designed to align with the boxe's. The result of the PCB design is shown in the figure below:

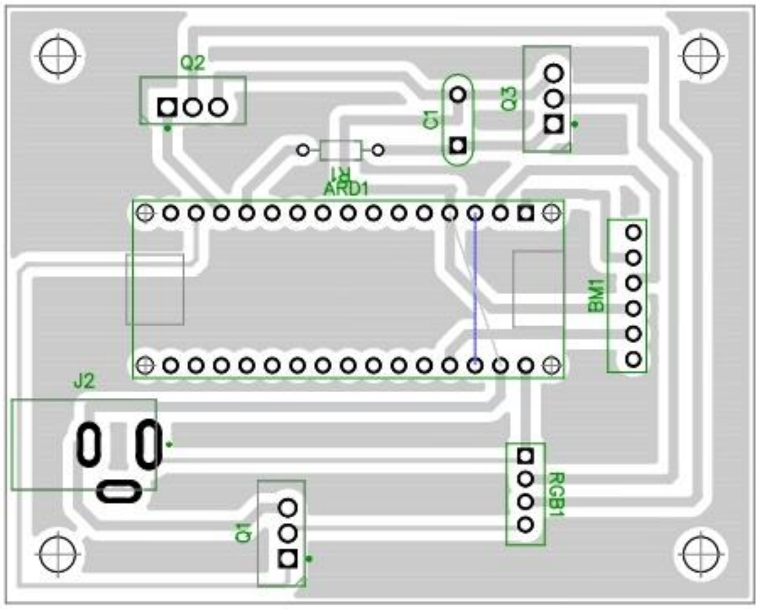


Figure 35: Image showing the PCB layout.

To ensure that the design was perfect and without any errors, verification of the PCB layout was done by using the check design rules and check net connectivity feature of the DipTrace. Next, the Gerber and drill files were zipped together so that they could be viewed in an online Gerber viewer. This is to ensure that all layers are on top of each other and to spot and fix any errors before printing the PCB. The following images show the PCB layout on the Gerber viewer.

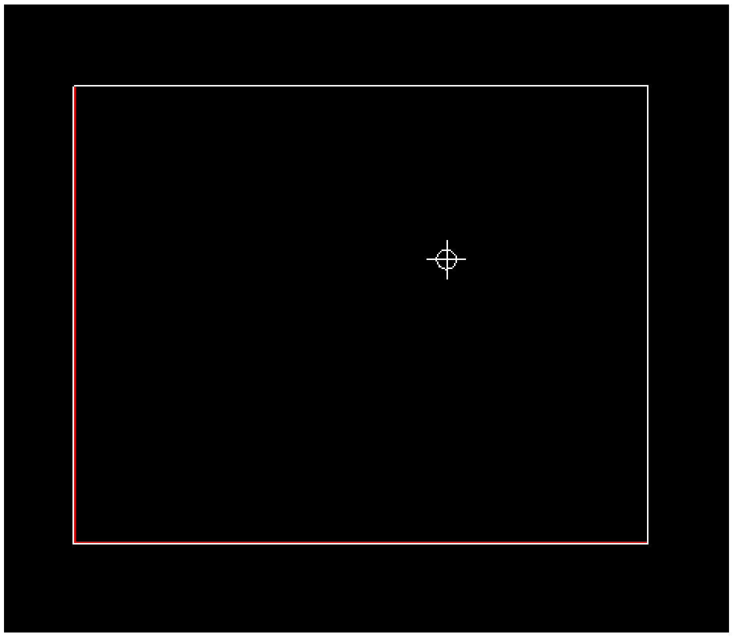


Figure 36: Image showing the rectangular dimensions of the PCB.

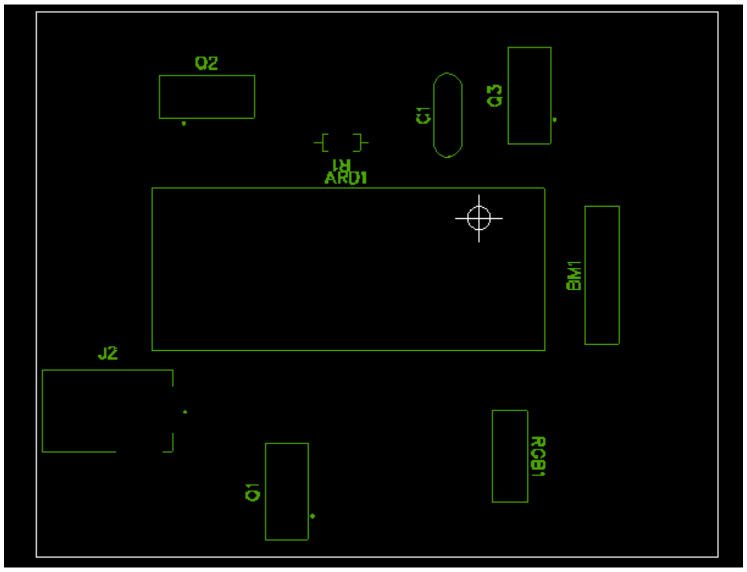


Figure 37: Image showing the components on the PCB.

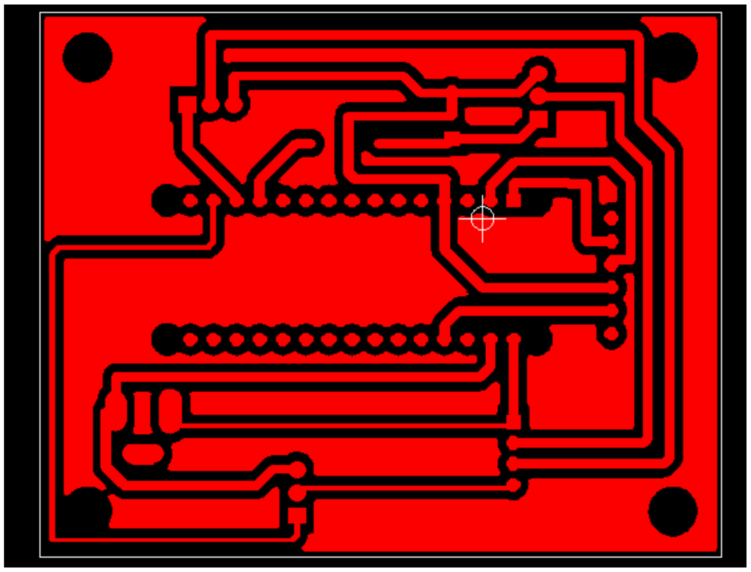


Figure 38: Image showing the traces of the PCB.

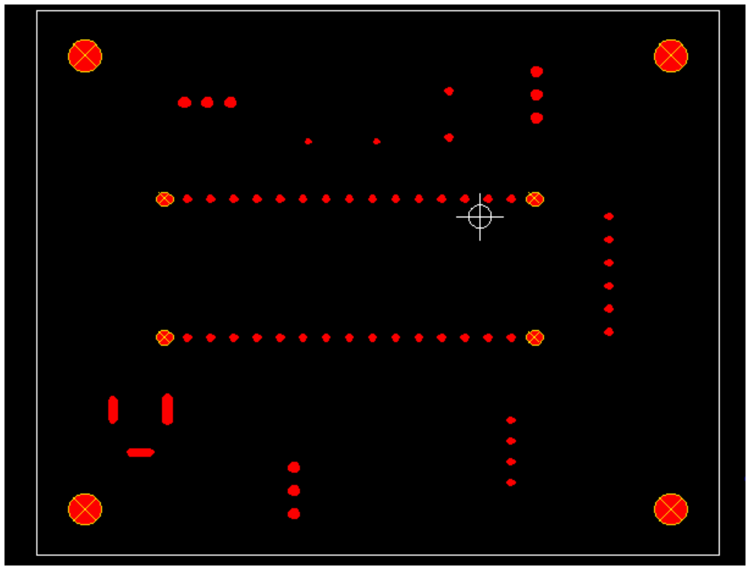


Figure 39: Image showing the drills on the PCB.

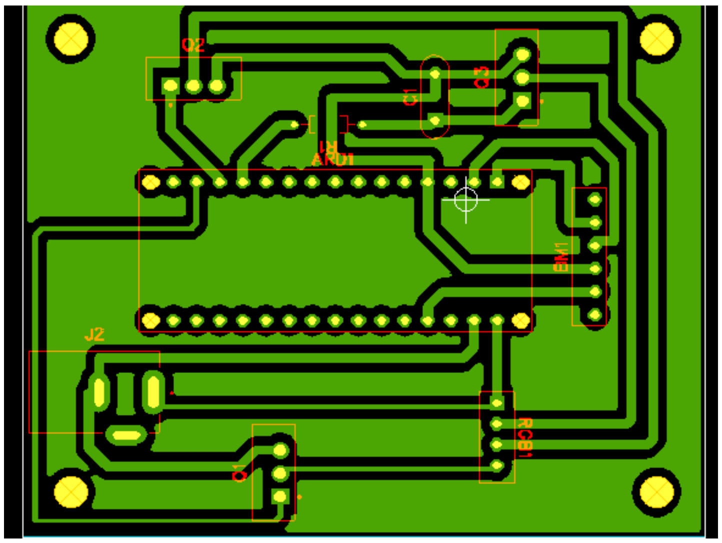


Figure 40:Image showing all the layers superimposed on each other.

After concluding that the PCB layout is the most optimum and error free, it was printed out. The printed PCB of this PCB design is shown below:

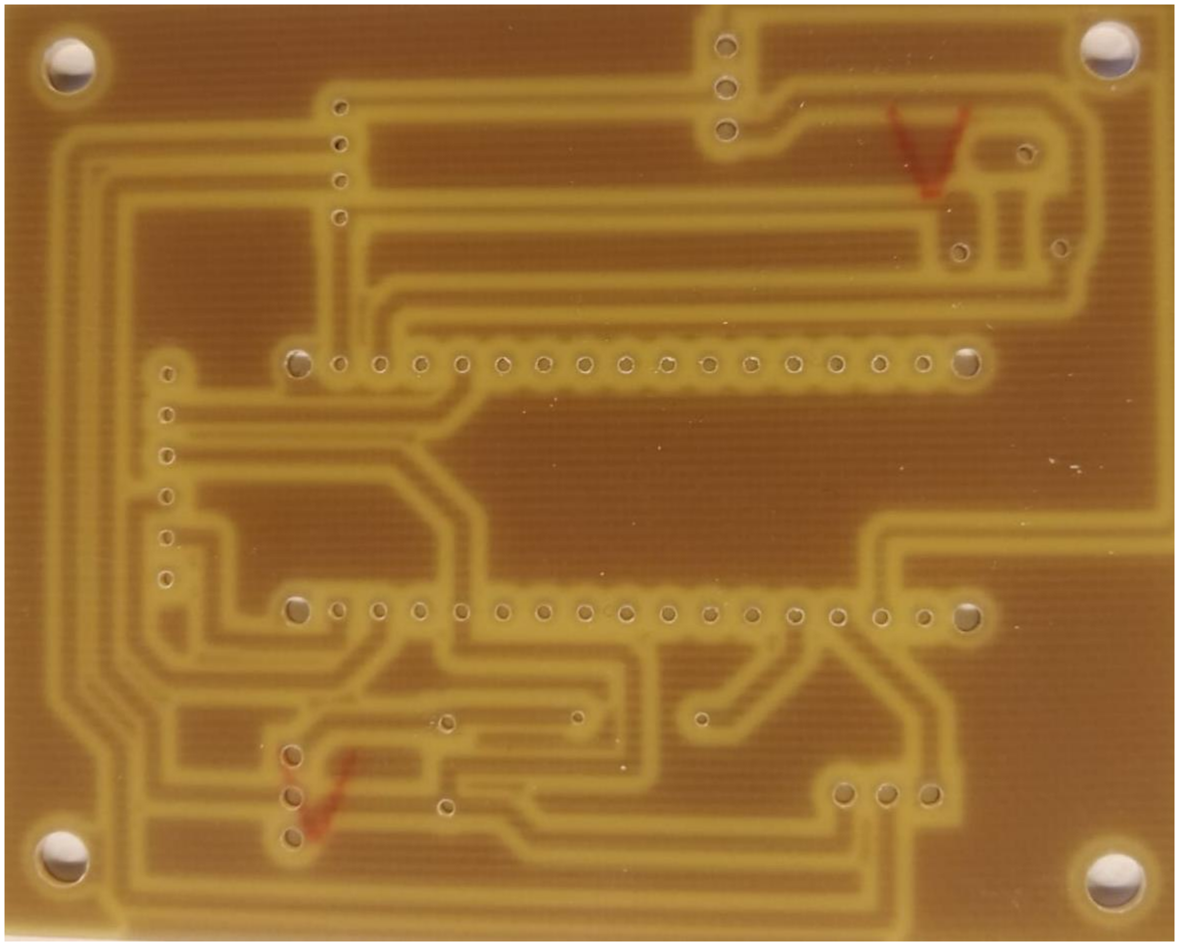


Figure 41: Image showing the top of the PCB.

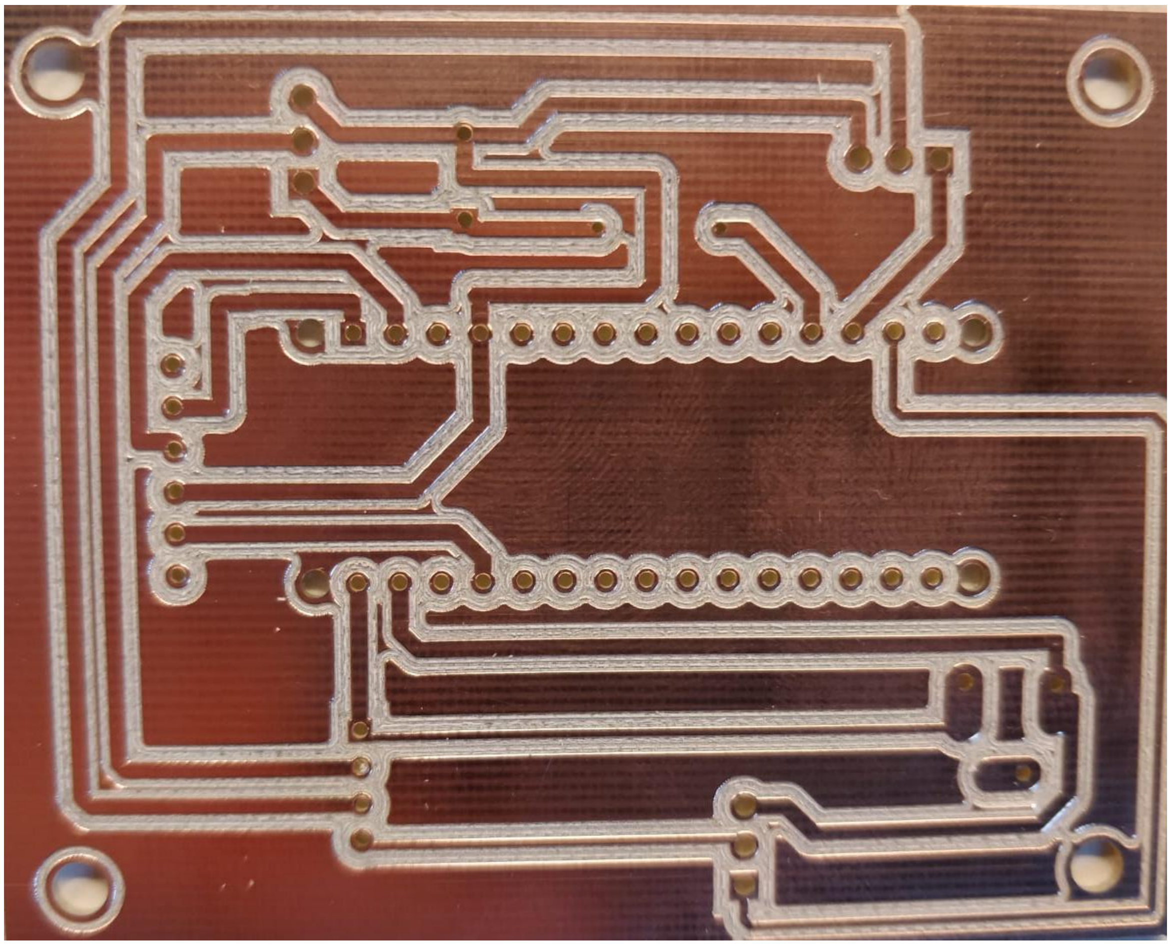


Figure 42: Image showing the bottom of the PCB.

It is important to highlight how the printed PCB had circular holes instead of oval holes. This meant the DC power jack could not fit in the PCB as is and some mechanism had to be developed to connect it to the circuit. To connect the power jack to the circuit, I soldered copper wires to the pins of the power jack and then soldered them to the PCB as well. Even though this was a quick fix to the problem, the power jack was very fragile and disconnected from the circuit twice, which had to soldered back again.

Next, the components are soldered on the PCB and screwed into the box. The following images show the black box implementation of the circuit:

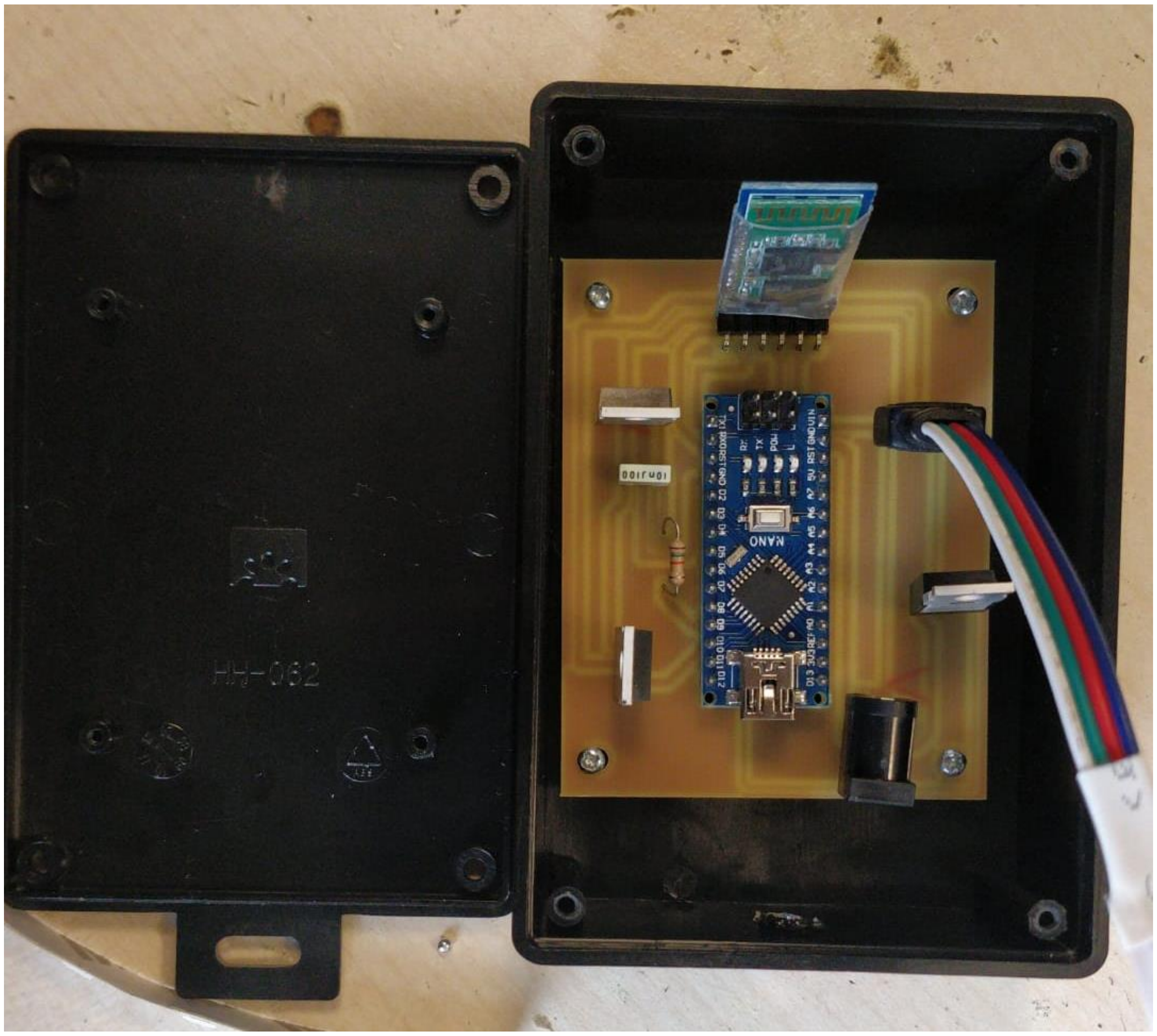


Figure 43: Image showing the circuit on a PCB inside the black box.

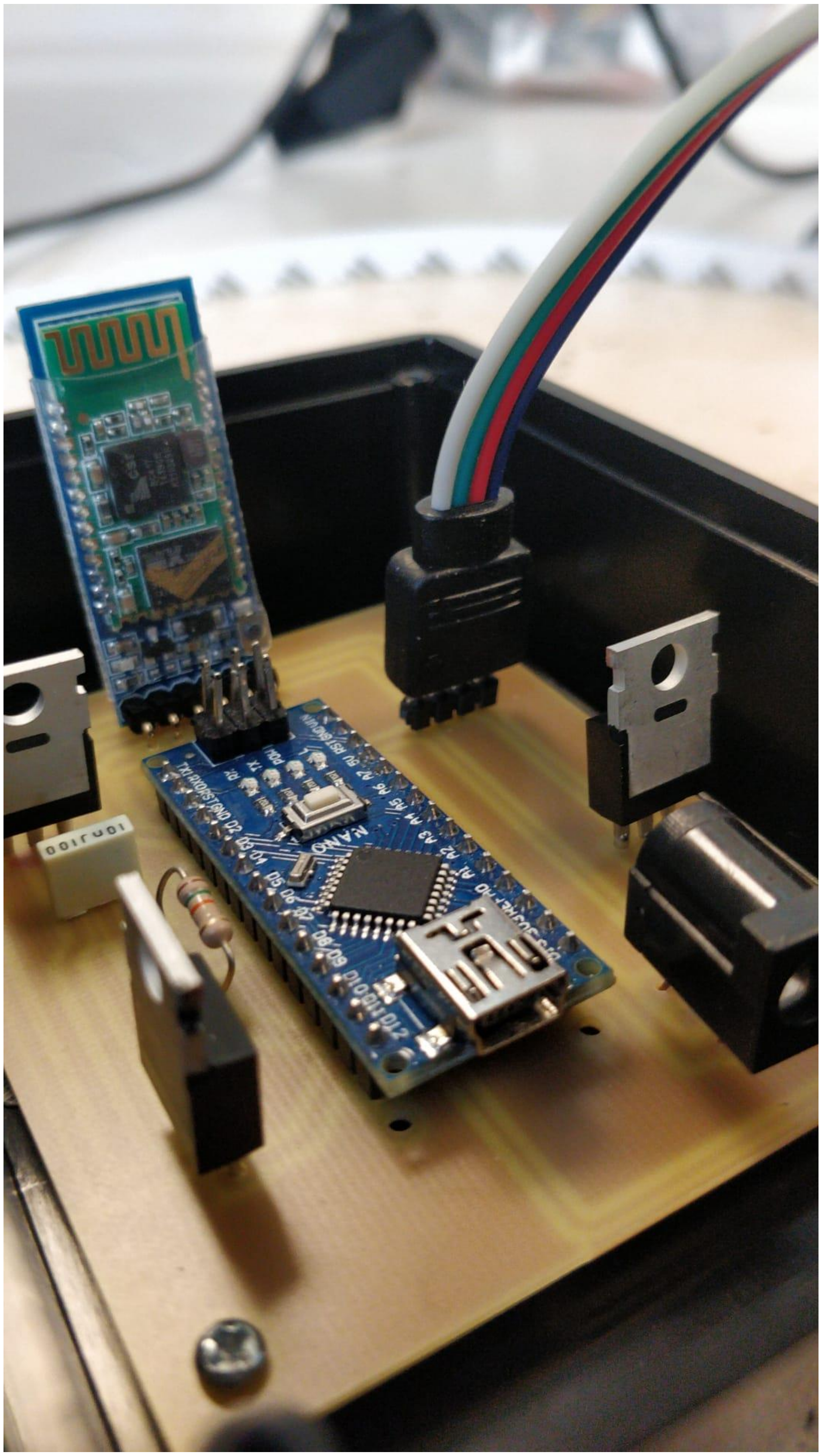


Figure 44: Image showing a close view of the PCB.

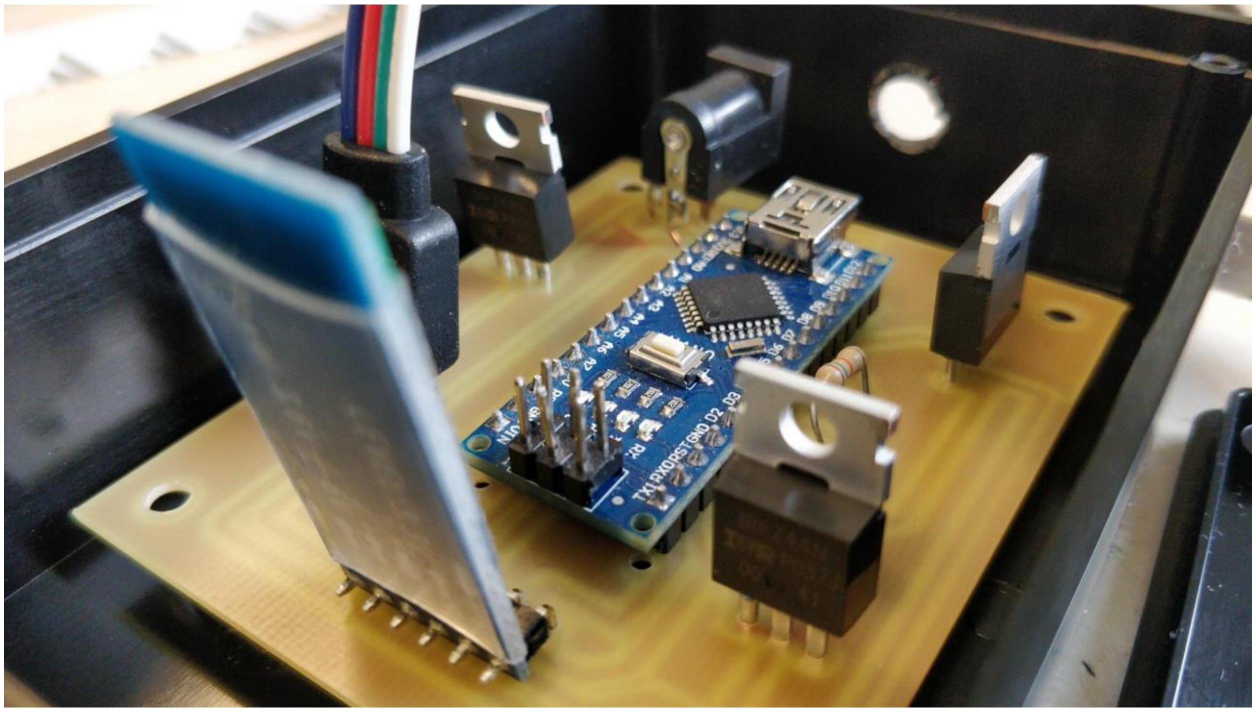


Figure 45: Another image of the PCB. This particular image highlights the hole created in the black box to ensure that the lid can be covered, and the power supply can still be connected to the power jack via this hole.

After putting everything together, the circuit is thoroughly tested and then demonstrated to the T.A. to get the term project check. This marked the end of the RGB Bluetooth controlled term project.

Conclusion

The term project has proven to be demanding and educating as well. Even though I had no prior knowledge about the Arduino Nano, C language, Bluetooth modules, and RGB LED strip, thanks to this project I now fully understand how they operate and why. I also relished the opportunity to design the circuit from scratch. By asking relevant questions and seeking answers by incorporating more components, I was able to incorporate the NMOS transistors in the saturation region and build something which has commercial value as well. It is important to highlight that initially I thought that the output of the Arduino using PMW is a DC value. However, the PMW is a square pulse (oscillating between voltage levels 0V and 5V). This meant the transistor at best would be continuously turning off and turning on in SAT region. To ensure that the circuit was always in SAT region, I incorporated a low-pass filter into my circuit (on the suggestion of Dr. Yusuf Ziya Ider). Even though the hardware implementation on the breadboard was simple, the PCB implementation was equally difficult. Initially, I had planned to run the fading lights code on the PCB implementation. After testing I tried to reprogram the Arduino to accommodate the Bluetooth feature. Because the Rx and Tx pins of the Arduino were connected to HC-05 via traces in the PCB, the program failed to upload on the ATMEL chip. To ensure the successful uploading of the code to Arduino, the Rx and Tx pins are required to be free as they are used in the process. Therefore, I had to desolder the HC-05 from the PCB. In the process, the PCB was slightly damaged. I realized that this problem could have been avoided had I used a voltage regulator to power the HC-05 instead of the 5V from the Arduino. By disconnecting the power supply and using the micro usb to upload the program, the Rx and Tx pins would be free as the HC-05 would be powered off. To verify that the circuit was operating as I had designed, I passed it through rigorous testing again on the breadboard using a spare Arduino Nano. However, the circuit did not work when implemented on the PCB as the mobile application gave the 'Broken Pipe' error. Upon inspection I realized that the Arduino used in the breadboard implementation was a clone and the mobile application had been designed for Arduino clones specifically. Therefore, I had to remove the original Arduino from the PCB and install the clone one. After this, the circuit worked as expected. I also realized that incorporating jumper wires as part of the PCB would have proven to be a daunting task. There is always a possibility that the jumper wires are loosely connected or perhaps that during the soldering phase the component to which the jumper wire is attached might be damaged. I saw many circuits in the lab of fellow peers fall victim to these issues. Consequently, I now recognize the importance of a good PCB design even better than before. Indeed, a good PCB design can save one from troubleshooting countless hours. I also realized the importance of correct patterns in the PCB design. Some of the patterns that I designed myself were not very precise. By adjusting the pins of the components, however, I was able to install them on the PCB with ease. Had the measurements been more inaccurate, I would have failed to install the components on the breadboard as is. Moreover, I understood the importance of recognizing PCB design constraints. Even though I had used the DC power jack sketch from the manufacturers, the printed PCB's drills were different. Upon speculation I learned that the PCB printing device in the EE 102 lab is only capable of drilling circular holes. This meant that now I had to make use of copper wires and somehow connect the power jack to the PCB. Consequently, a very fragile power jack was made in the end. During the testing phase, the power jack disconnected two times and had to be reconnected which damaged the PCB tracks as well. Had I learned about these design constraints in advance, I would have simply used bigger holes to accommodate the power jack.

In short, the project has proven to be a great success. By leveraging knowledge acquired in electronic circuit design, microcontrollers, and analog electronics, I was able to put together the transistors and Arduino to produce a commercializable project: Bluetooth Controlled RGB LED Strip.

References

[1] Android Controller RBG LED Strip, instructables. Accessed on: March 12, 2021. [Online]. Available: https://www.instructables.com/Android-Controlled-RGB-LED-Strip-Using-Arduino/

[2] Hc-05 Bluetooth module, components101. Accessed on: April 01, 2021. [Online]. Available: https://components101.com/wireless/hc-05-bluetooth-module

[3] Arduino Nano R3 clone, open circuit. Accessed on: March 12, 2021. [Online]. Available: https://opencircuit.shop/Product/Arduino-Nano-R3-clone

[4] Arduino Nano Pin, schematic, and specification in great detail, electrophiles. Accessed on: April 01, 2021. [Online]. Available: https://www.etechnophiles.com/arduino-nano-pinout-schematic-and-specifications-in-detail/

[5] PMW, Arduino. Accessed on: April 19, 2021. [Online]. Available: https://www.arduino.cc/en/Tutorial/Foundations/PWM

[6] LED Light Strip 12V 24V RGB RGBW RGBWW PC SMD 5050 60Led / s 5 M 12 24 V Volt LED Strip Lights Waterproof Lamp Ribbon TV Backlight, AliExpress. Accessed on: April 19, 2021. [Online]. Available: https://tr.aliexpress.com/item/32955763018.html

[7] Data Sheet 5050 SMD 60 LED/m Indoor Strip LED, ilker. Accessed on: April 19, 2021. [Online]. Available: https://www.iled.com/class/INNOVAEditor/assets/gallery2/5100-5105.pdf

[8] Mosfet Amplifier, electronics tutorials. Accessed on: March 12, 2021. [Online]. Available: https://www.electronics-tutorials.ws/amplifier/mosfet-amplifier.html

[9] Android Controlled RGB LED Strip Using Arduino, huckster.io. Accessed on: June 25, 2021. [Online]. Available: https://www.hackster.io/dhritimanhb2015/android-controlled-rgb-led-strip-using-arduino-f9a9b3#toc-introduction-0

Appendix

Fading LEDs

```
#define green 9
#define red 10
#define blue 11
int wait = 50;
int brightnessGreen = 0;
int brightnessRed = 0;
int brightnessBlue = 0;
void setup()
{

}

void loop()
{
  analogWrite(green, brightnessGreen); //Feeding brightness value to green pin 9
  analogWrite(red, brightnessRed);
  analogWrite(blue, brightnessBlue);
  delay(wait);
//Sweep through all Brightnesses of Green while brightnesses of red and blue remain same
for (brightnessGreen = 0; brightnessGreen < 255; brightnessGreen++) {
  analogWrite(green,brightnessGreen);
  delay(wait);
}

//Sweep through all Brightness of Red while brightnesses of green and blue remain same
for (brightnessRed = 0; brightnessRed < 255; brightnessRed++) {
  analogWrite(red,brightnessRed);
  delay(wait);
}

//Sweep through all Brightness of Blue while brightnesses of red and green remain same
for (brightnessBlue = 0; brightnessBlue < 255; brightnessBlue++) {
  analogWrite(blue,brightnessBlue);
  delay(wait);
}

for (brightnessRed = 255; brightnessRed > 0; brightnessRed--) {
  analogWrite(blue,brightnessBlue);
  delay(wait);
}
}
```

Bluetooth Communication

```
#include <SoftwareSerial.h>

SoftwareSerial BLU(0,1);

#define greenPin 9
#define redPin 10
#define bluePin 11

void setup()
{
  //Serial setup
  Serial.begin(9600);
  Serial.println("-= HC-05 Bluetooth RGB LED =-");
  BLU.begin(9600);
  BLU.println("-= HC-05 Bluetooth RGB LED =-");

  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

  setColor(255, 0, 0);
  delay(500);
  setColor(0, 255, 0);
  delay(500);
  setColor(0, 0, 255);
  delay(500);
  setColor(255, 255, 255);
}

void loop()
{
  while (BLU.available() > 0)
  {
    int redInt = BLU.parseInt();
    int greenInt = BLU.parseInt();
    int blueInt = BLU.parseInt();

    redInt = constrain(redInt, 0, 255);
    greenInt = constrain(greenInt, 0, 100);
    blueInt = constrain(blueInt, 0, 255);

    if (BLU.available() > 0)
    {
      setColor(redInt, greenInt, blueInt);

      Serial.print("Red: ");
      Serial.print(redInt);
      Serial.print(" Green: ");
      Serial.print(greenInt);
      Serial.print(" Blue: ");
      Serial.print(blueInt);
      Serial.println();

      BLU.flush();
    }
  }
}

void setColor(int red, int green, int blue)
{
  analogWrite(redPin, red);
  analogWrite(greenPin, green);
  analogWrite(bluePin, blue);
}
```