Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

РАЗРАБОТКА СХЕМЫ УПРАВЛЕНИЯ 3D КУБОМ ИЗ СВЕТОДИОДОВ

Студент ИУ8-63 _____ Горбачев А.А.
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ Усанов А.Е.
(Подпись, дата) (И.О.Фамилия)

2021 г.

Оглавление

Введение

Предметная область курсового проекта включает в себя проектирование, и конструирование светодиодного куба, а также разработка программы прошивки для платформы Arduino и подготовка к демонатрации проекта.

3D LED-cube может отображать объемные изображения на скорости 30 кадров в секунду. Куб представляет собой сетку из светодиодов, соединенных между собой.

Область применения LED куба: на выстовачных стендах в качестве основного элемента, в клубах, барах, на концертах и на других массовых мероприятиях.

Такие конструкции предназначены для вывода красочных объемных изображений, которые позволяют привлекать внимание людей. Поэтому было принято решение разработать инсталляционный светодиодный куб, чтобы добиться актуальности данного проекта на сегодняшний день.

Цель работы

Целью данной работы является получение практических навыков в области разработки светодиодного куба размерности 4x4x4 и реализации программы прошивки для платы Arduino Nano на базе микропроцессора Atmega328P.

Задачи

В данной работе можно выделить следующие поставленные задачи, необходимые для достижения поставленной цели:

1) Выполнить анализ требований и синтезировать принципиальную схему;
2) Выполнить разводку для последующего изготовления печтаной платы;
3) Изготовить печатную плату методом ЛУТ;
4) Произвести монтаж элементов на печатную плату;
5) Разработать программу прошивки для Arduino Nano;
6) Протестировать собранную конструкцию.

Ход работы

Теоретическая часть

Последовательный – параллельный ввод данных

74HC595 — восьмиразрядный сдвиговый регистр с последовательным вводом, последовательным или параллельным выводом информации, с триггером-защёлкой и тремя состояниями на выходе. Другими словами этот регистр позволяет контролировать 8 выходов, используя всего несколько выходов на самом контроллере. При этом несколько таких регистров можно объединять последовательно для каскадирования. Распиновка контактов сдвигового регистра приведена в таблице 1.

74HC595 может отдавать сигналы не только параллельно, но и последовательно [1]. Это необходимо при объединении нескольких регистров, для получения 16 и более выходов. В этом случае первые 8 бит сигнала передаются на следующий регистр для параллельного вывода на нём.

Подключенный к контроллеру Arduino Nano, сдвиговый регистр будет обеспечивать параллельный вывод информации для управление транзисторным затвором.

С выхода D13 платы Arduino будет подаваться тактовый сигнал для записи и вывода данных со сдвигового регистра. Контакт D11 используется для записи данных в регистр, а D10, соответственно, управляет защелкой.

Таблица 1. Распиновка контактов сдвигового регистра HC595

74HC595		
Пины 1-7, 15	Q1 – Q7, Q0	Параллельные выходы
Пин 8	GND	Земля
Пин 9	Q7”	Выход для последовательного соединения регистров

Пин 10	MR	Сброс значений регистра. Сброс происходит при получении LOW
Пин 11	SH_CP	Вход для тактовых импульсов
Пин 12	ST_CP	Защелка. Синхронизация выходов
Пин 13	OE	Вход для переключения состояния выходов из высокоомного в рабочее
Пин 14	DS	Вход для последовательных данных
Пин 16	Vcc	Питание

Платформа Arduino

Arduino — это электронный конструктор и удобная платформа быстрой разработки электронных устройств. Платформа пользуется огромной популярностью во всем мире благодаря удобству и простоте языка программирования, а также открытой архитектуре и программному коду. Устройство программируется через USB без использования программаторов. [2]

Arduino позволяет компьютеру выйти за рамки виртуального мира в физический и взаимодействовать с ним. Устройства на базе Arduino могут получать информацию об окружающей среде посредством различных датчиков, а также могут управлять различными исполнительными устройствами.

Основная особенность Arduino заключается в том, что существует большое количество библиотек, которые автоматизируют большинство прикладных задач. Используемые микроконтроллеры в Arduino уже имеют прошитый загрузчик, поэтому программатор не нужен, достаточно соединить плату с компьютером через USB или переходник UART-USB, и загрузить программу.

Имеется на плате и возможность прошить загрузчик в микроконтроллер самостоятельно с помощью программатора, в Arduino IDE встроена поддержка

наиболее популярных дешевых программаторов, есть штыревой разъем для внутрисхемного программирования (ICSP для AVR, JTAG для ARM).

В большинстве устройств Ардуино используются микроконтроллеры Atmel AVR ATmega328, ATmega168, ATmega2560, ATmega32U4, ATTiny85 с частотой тактирования 16 или 8 МГц. Есть также платы на процессоре ARM Cortex M. [3]

Платформа Nano, построенная на микроконтроллере ATmega328 (Arduino Nano 3.0) или ATmega168 (Arduino Nano 2.x), имеет небольшие размеры и может использоваться в лабораторных работах. Отличие заключается в отсутствии силового разъема постоянного тока и работе через кабель Mini-B USB. Краткие характеристики Arduino Nano 3.0, представлены в таблице 2 и на рисунках 1 и 2.

Таблица 2. Характеристики Arduino Nano 3.0

Микроконтроллер	Atmel ATmega168 или ATmega328
Рабочее напряжение (логическая уровень)	5 В
Входное напряжение (рекомендуемое)	7-12 В
Входное напряжение (предельное)	6-20 В
Цифровые Входы/Выходы	14 (6 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	8
Постоянный ток через вход/выход	40 мА
Флеш-память	16 Кб (ATmega168) или 32 Кб (ATmega328) при этом 2 Кб используются для загрузчика
ОЗУ	1 Кб (ATmega168) или 2 Кб (ATmega328)
EEPROM	512 байт (ATmega168) или 1 Кб (ATmega328)
Тактовая частота	16 МГц
Размеры	1.85 см x 4.2 см

Микроконтроллер ATmega168 имеет 16 кБ флеш-памяти для хранения кода программы, а микроконтроллер ATmega328, в свою очередь, имеет 32 кБ (в обоих случаях 2 кБ используется для хранения загрузчика). ATmega168 имеет 1 кБ ОЗУ и 512 байт EEPROM (которая читается и записывается с помощью библиотеки EEPROM), а ATmega328 – 2 кБ ОЗУ и 1 Кб EEPROM. [4]

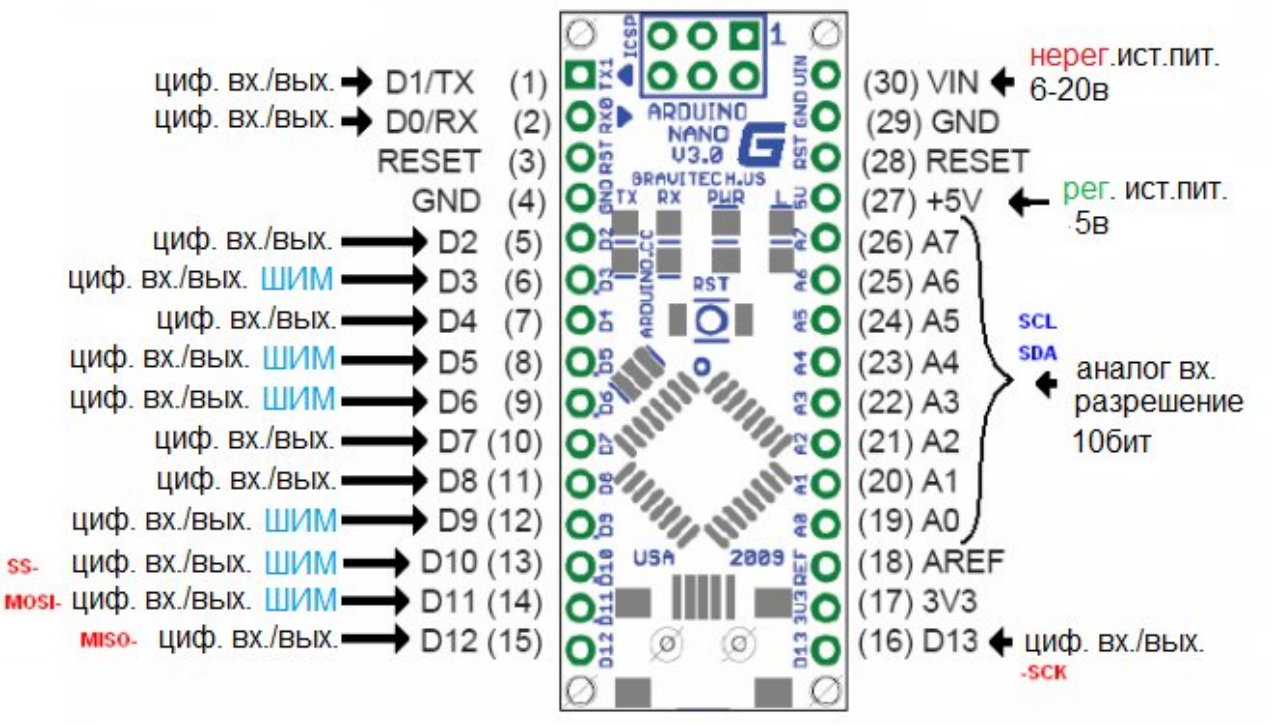


Рисунок 1 – Обозначения портов ввода/вывода на плате Arduino Nano

Pin No.	Name	Type	Description
1-2, 5-16	D0-D13	I/O	Digital input/output port 0 to 13
3, 28	RESET	Input	Reset (active low)
4, 29	GND	PWR	Supply ground
17	3V3	Output	+3.3V output (from FTDI)
18	AREF	Input	ADC reference
19-26	A0-A7	Input	Analog input channel 0 to 7
27	+5V	Output or Input	+5V output (from on-board regulator) or +5V (input from external power supply)
30	VIN	PWR	Supply voltage

Рисунок 2 – Распиновка контактов Arduino Nano

Каждый из 14 цифровых выводов Nano (рисунок 1), используя функции pinMode, digitalWrite, и digitalRead, может настраиваться как вход или выход. Выводы работают при напряжении 5 В. Каждый вывод имеет нагрузочный резистор (стандартно отключен) 20-50 кОм и может пропускать до 40 мА. Некоторые выводы имеют особые функции [3]:

– последовательная шина: 0 (RX) и 1 (TX). Выводы используются для получения (RX) и передачи (TX) данных TTL. Данные выводы подключены к соответствующим выводам микросхемы последовательной шины FTDI USB-to-TTL;

– внешнее прерывание: 2 и 3. Данные выводы могут быть сконфигурированы на вызов прерывания либо на младшем значении, либо на переднем или заднем фронте, или при изменении значения;

– ШИМ: 3, 5, 6, 9, 10, и 11. Любой из выводов обеспечивает ШИМ с разрешением 8 бит при помощи функции analogWrite;

– SPI 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Посредством данных выводов осуществляется связь SPI, которая, хотя и поддерживается аппаратной частью, не включена в язык Arduino;

– LED 13. Встроенный светодиод, подключенный к цифровому выводу 13. Если значение на выводе имеет высокий потенциал, то светодиод горит;

– I2C: A4 (SDA) и A5 (SCL). Посредством выводов осуществляется связь I2C (TWI). Для создания используется библиотека Wire.

Загрузка скетча на плату Arduino

В Ардуино IDE компиляция скетча начинается при нажатии кнопки Verify, после этого скетч может быть загружен в память Ардуино. Перед загрузкой скетча необходимо указать порт, к которому подключена плата Arduino в меню «Инструменты» - «Порт». В Windows такими портами выступают COM1 или COM2 для плат последовательной шины. COM4, COM5, COM7 и выше для плат USB. Далее выбирается устройство в меню «Инструменты» - «Плата». А также необходимо указать тип процессора в меню «Инструменты» - «Процессор».

После выбора порта и платформы необходимо нажать кнопку загрузки на панели инструментов или выбрать пункт меню «Скетч» - «Загрузка». Современные платформы Arduino перезагружаются автоматически перед загрузкой. На старых платформах необходимо нажать кнопку перезагрузки. На большинстве плат во время процесса будут мигать светодиоды RX и TX, обозначая, что происходит обмен данными. Среда разработки Arduino выведет сообщение об окончании загрузки или об ошибках.

Загрузка происходит напрямую в память устройства. На микроконтроллере Ардуино имеется 3 вида памяти – флеш-память, которая используется для хранения скетчей, ОЗУ для хранения переменных и EEPROM для хранения постоянной информации. Из этих типов памяти флеш-память и EEPROM являются энергонезависимыми, то есть информация сохраняется при выключении питания. ОЗУ используется только для хранения данных, которые имеют отношение к исполняемой программе.

На объем памяти не влияют размер имени переменных и комментарии. Компилятор устроен таким образом, что не включает эти данные в скомпилированный скетч.

Для измерения объема занимаемой памяти ОЗУ используется скетч из библиотеки MemoryFree [5]. В ней имеется специальная функция freeMemory,

которая возвращает объем доступной памяти. Также эта библиотека широко используется для диагностики проблем, которые связаны с нехваткой памяти.

Флеш память является безопасным и удобным способом хранения данных, но некоторые факторы ограничивают ее использование. Для флеш-памяти характерна запись данных блоками по 64 байта. Также флеш-память гарантирует сохранность информации для 100000 циклов записи, после чего информация искажается. Во флеш-памяти имеется загрузчик, который нельзя удалять или искажать. Это может привести к разрушению самой платы.

EEPROM память используется для хранения всех данных, которые потребуются после отключения питания. Для записи информации в EEPROM нужно использовать специальную библиотеку EEPROM.h, которая входит в число стандартных библиотек в Arduino IDE. Чтение и запись информации в EEPROM происходит медленно, порядка 3 мс. Также гарантируется надежность хранения данных для 100000 циклов записи, потому лучше не выполнять запись в цикле. [3]

Разработка и проектирования светодиодного куба для Arduino Nano

Проектирование платы управления

Для того, что бы обеспечить подсветку всех 64 светодиодов, за неимением такого большого количества выводов на плате Arduino Nano, воспользуемся сдвиговыми регистрами. При наличии четырех рядов с четырьмя светодиодами в каждом, понадобится 16 бит выходных данных, соответственно. Следовательно, два сдвиговых регистра будут обеспечивать управление всеми четырьмя рядами. При этом все три сдвиговых регистра объединены в каскад.

Два сдвиговых регистра будут выводить 16 бит данных для 16 анодов светодиодов, проходя через резисторы 220 Ом, расположенных на нижнем слое. Третий сдвиговый регистр будет отвечать за подачу сигнала на базу транзистора, для открытия эмиттера на землю с катодов светодиодов, которые идут на коллектор транзистора (Рисунок 3). Таким образом, получим динамическую индикацию с общим анодом. [6]

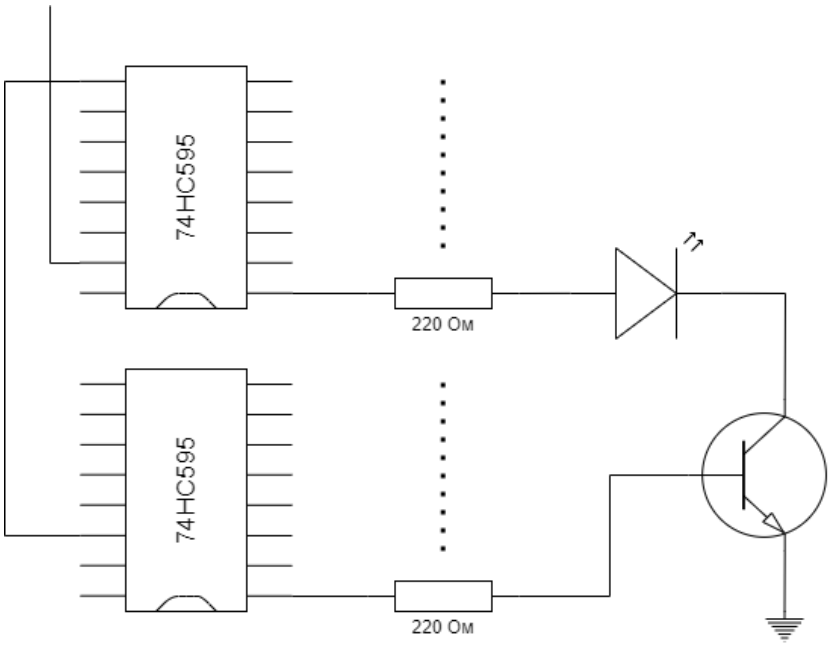


Рисунок 3 – Схема динамической индикации на сдвиговых регистрах и NPN транзисторе

Для управления режимами подсветки используются две тактовые кнопки, подключенные к аналоговым входам A2 и A3, два светодиода, для индикации режима загрузки, подключенные к входам A0 и A1 через 2 резистора 220 Ом, а также LCD дисплей для отображения выбранного режима подсветки, подключенного через I2C к контактам A4 и A5 на плате Arduino Nano.

Так как питание на плату будет поступать от блока питания, подключенного в сеть 220В, для этого необходимо установить на плате конденсатор на 1000uF для компенсирования скачков напряжения, тем самым обезопасив контроллер Arduino от возможного выхода из строя.

Светодиоды будут потреблять 64 * 0.009 = 0.576 Ампер, но так как транзисторы обеспечат нам динамическую светодиодную индикацию, то полученное число еще поделим на 4 и получим 0.144 Ампер. Таким образом, выходного тока с блока питания будет вполне достаточно для того, чтобы питать всю плату.

Составим принципиальную схему в EasyEDA [7] результат приведен в приложении А.

В этой же среде выполним разводку платы в режиме PCB (приложение Б и В), минимизировав количество соединений на верхнем слое, из-за того, что имеющийся текстолит односторонний.

Прошивка Arduino Nano

Для прошивки платы Arduino была написана программа, приведенная в приложении Г. Основными функциями для работы прошивки, выступаю void loop() и void setup().

Функция setup() вызывается, когда стартует скетч. Используется для инициализации переменных, определения режимов работы выводов, запуска используемых библиотек и т.д. Функция setup запускает только один раз, после каждой подачи питания или сброса платы Arduino.

После вызова функции setup(), которая инициализирует и устанавливает первоначальные значения, функция loop() вызывается бесконечное количество раз в цикле, позволяя программе совершать вычисления и реагировать на них. Используется для активного управления платой Arduino.

Так как Arduino взаимодействует со сдвиговыми регистрами, то для этого используется библиотека SPI [8], позволяющая осуществлять последовательную подачу данных и параллельный вывод их с выходов сдвиговых регистров.

Так как сдвиговых регистра три, объединенных в каскад, то для начала через канал SPI подается байт, содержащий только одну «1» , который пойдет на сдвиговый регистр, отвечающий за подачу сигналов на базу транзисторов, тем самым открыв определенный транзисторный ключ, это сделано для выбора определенного слоя. Дальше подаются 8 бит для подсветки последних 2ух рядов и еще 8 бит для подсветки первых 2ух рядов, где «1» в наборе бит будет отвечать за подачу сигнала на определенные аноды светодиодов.

Данные для отображения хранятся в двумерном байтовом массиве следующим образом: байтовое значение cube[i][j], оно будет подаваться через канал SPI на сдвиговые регистры, где i – номер слоя, на котором будут подсвечиваться светодиоды, а j – номер ряда для отображения. Данные заносятся в массив с помощью побитовых операций.

Также в программе предусмотрены методы, которые позволяют выбирать режим работы программы changeMod(), а также методы, отвечающие за различные режимы подсветки.

За отображения данных на LCD дисплее отвечает библиотека LiquidCrystal_I2C.h [9]. Каждый раз, при смене режима, название режима на дисплее меняется, что делает взаимодействие со светодиодном кубом удобнее.

Сам контроллер подключается по USB к компьютеру, выбирается порт и процессор, после чего, с помощью программатора, скетч загружается на плату.

Компоненты светодиодного куба

Для того чтобы собрать LED-cube необходимы следующие компоненты:

- Светодиоды 64 шт.
- Резистор на 220 Ом 18 шт.
- Arduino Nano
- Сдвиговый регистр 74HC595 3 шт.
- Биполярный NPN транзистор TIP41C 4 шт.
- Светодиоды красного и зеленого свечения для индикации загрузки
- Кнопки тактовые 2 шт.
- Емкостный конденсатор 1000uF
- LCD дисплей 1602
- I2C для подключения LCD дисплея

Для изготовления платы необходимо следующее:

- Хлорид железа (III)
- Текстолит размера не менее 160 х 160 мм
- Пластиковый тазик
- Инструменты для пайки
- Провода с сечением не меньше 0.5 мм
- Термоусадочная трубка

Изготовление печатной платы

Для изготовления печатной используется метод лазерно-утюжной технологии (ЛУТ) [10]. Для этого необходимо распечатать схему на глянцевой бумаге, приложить лист к гладкой отшлифованной поверхности текстолита и, используя утюг, перенести рисунок на текстолит. После чего необходимо поместить плату в теплую воду и оставить ее на 10 минут. Затем аккуратно удаляется бумага. Если в процессе какие-то дорожки перенеслись плохо – можно дорисовать их перманентным маркером. Результат приведен на рисунке 4.

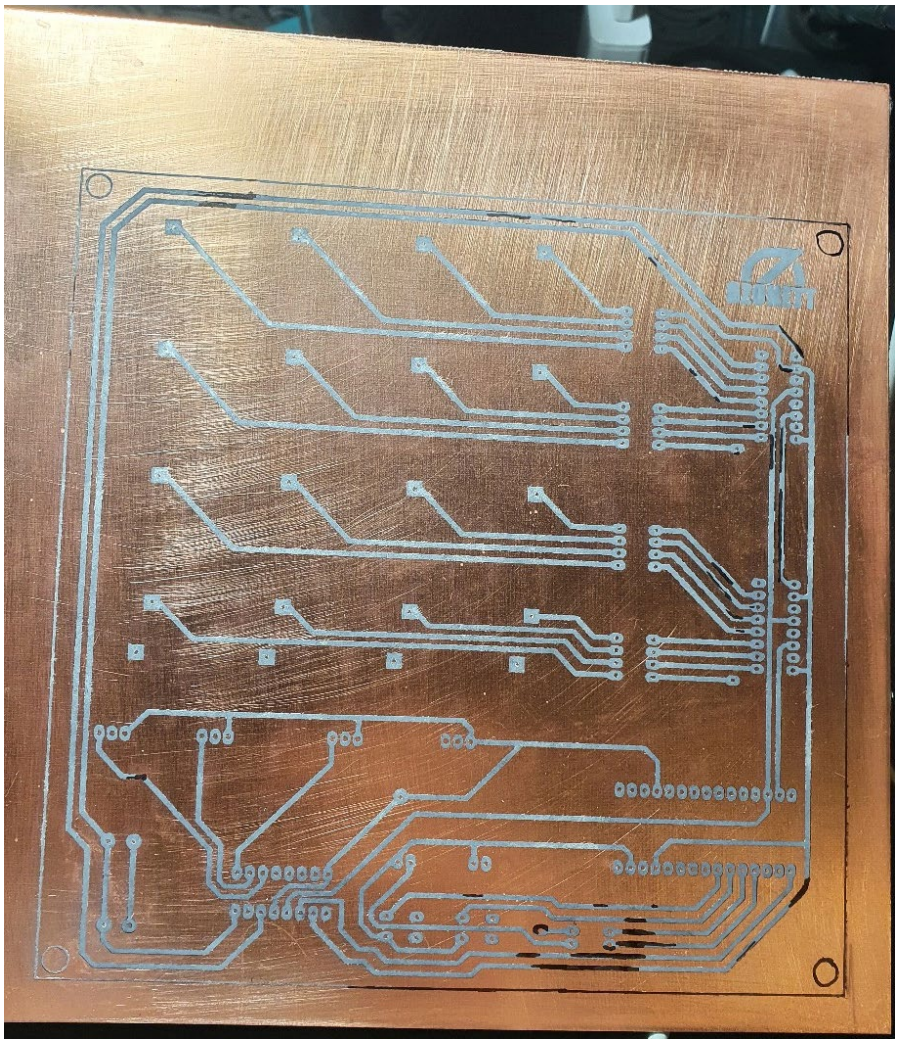


Рисунок 4 – Плата после переноса рисунка на текстолит

Теперь необходимо развести раствор хлорного железа и оставить плату травится в нем на протяжении 40 минут. Получим следующий результат (Рисунок 5)

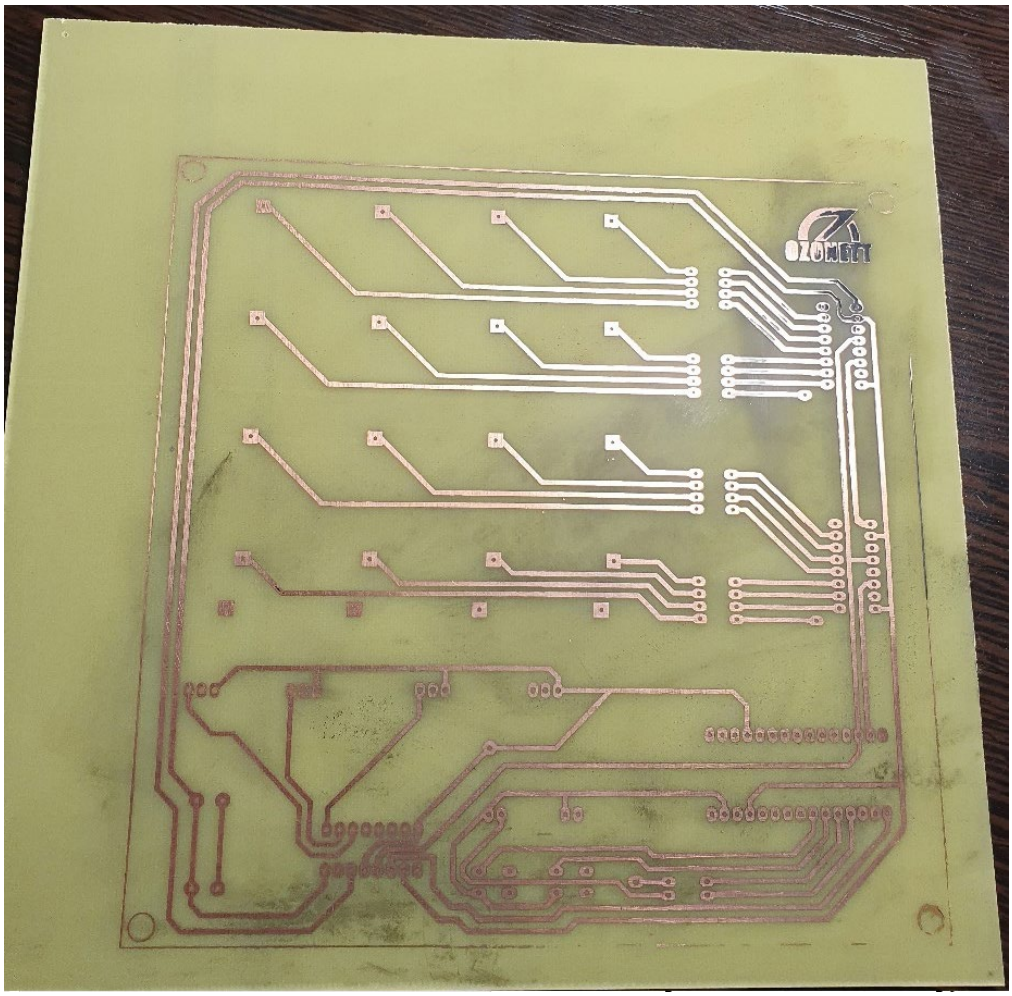


Рисунок 5 – Плата после травления в хлорном железе

После травления, необходимо залудить плату. Для этого флюс наносится по всей поверхности платы и оловянным припоем обрабатываются все медные участки платы. По окончанию работ, флюс необходимо отмыть спиртом или ацетоном.

Далее просверливаются все отверстия, для этого используется маленькая дрель со сверлом диаметра 0.9 мм.

Сборка светодиодного куба

Начать монтаж следует со сборки куба из светодиодов. Для того чтобы правильно собрать куб, необходимо понимать, что у светодиода длинная нога является анодом, т.е. плюсом, а короткая – катодом или минусом. Сборка слоев светодиодного куба производится, как показано на рисунке 6. Аноды паяются с анодами, а катоды, соответственно, с катодами.

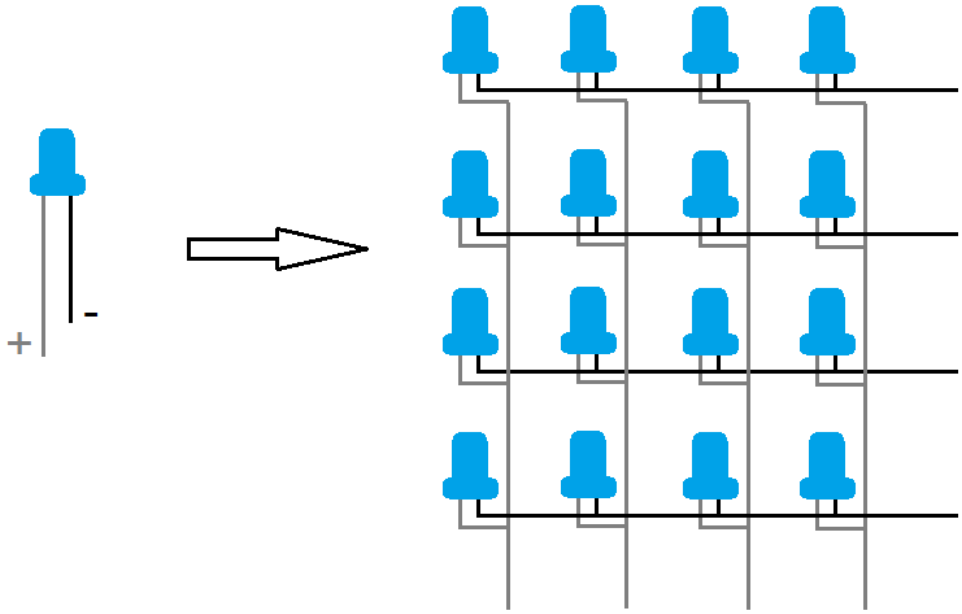


Рисунок 6 – Схема сборки слоя светодиодного куба

После того, как было произведено изготовление четырех таких слоев, необходимо припаять медную или другую проводящую проволоку на катоды для придания прочности конструкции и упрощению передачи сигналов как показано зелеными линиями на рисунке 7.

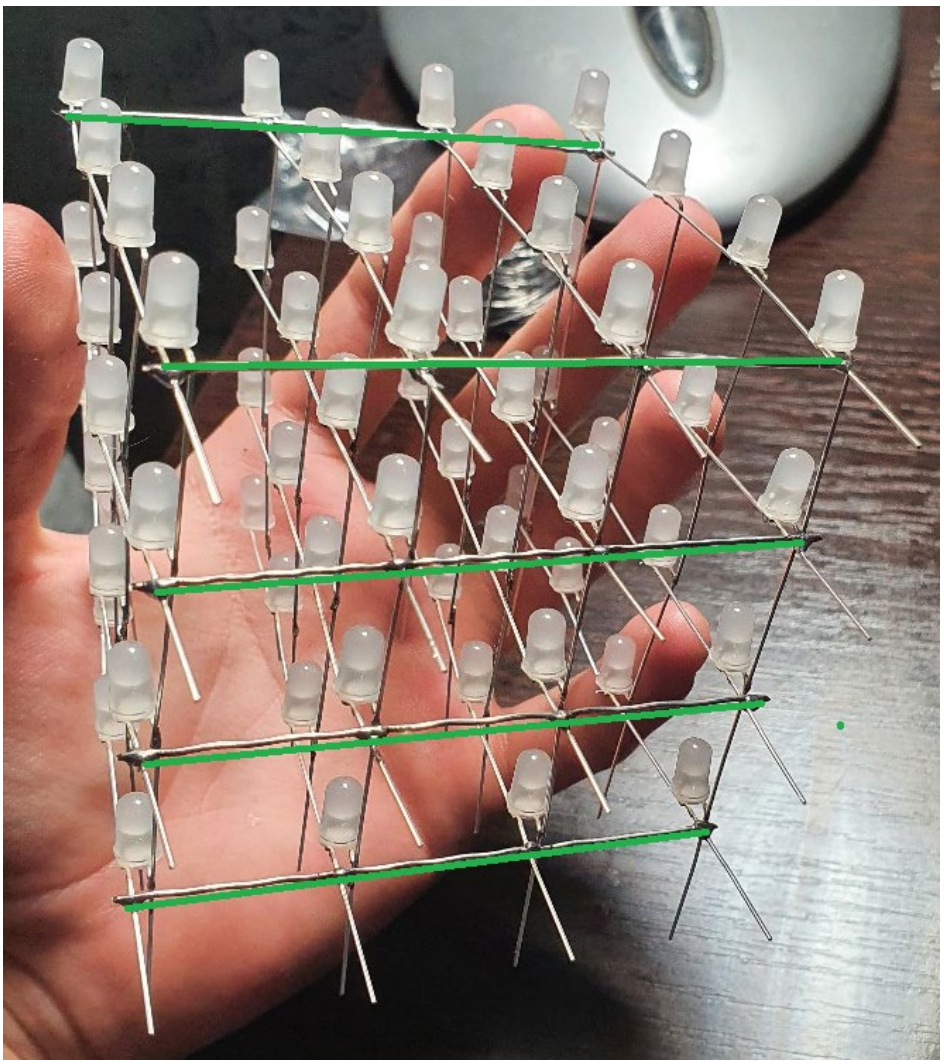


Рисунок 7 – Результат сборки куба из светодиодов

Монтаж элементов на плату

Теперь необходимо произвести монтаж компонентов на плату, согласно схемам печатной платы, как показано на верхнем и нижнем слое (приложение Б и В). Из-за того, что текстолит изначально был однослойным, то верхний слой будет реализован с помощью проводов, припаянных к соответствующим пинам.

Также к коллекторам биполярных транзисторов TIP41C припаяны медные проволки, идущие к слоям с катодами светодиодного куба как показано на рисунке 8 красными линиями.

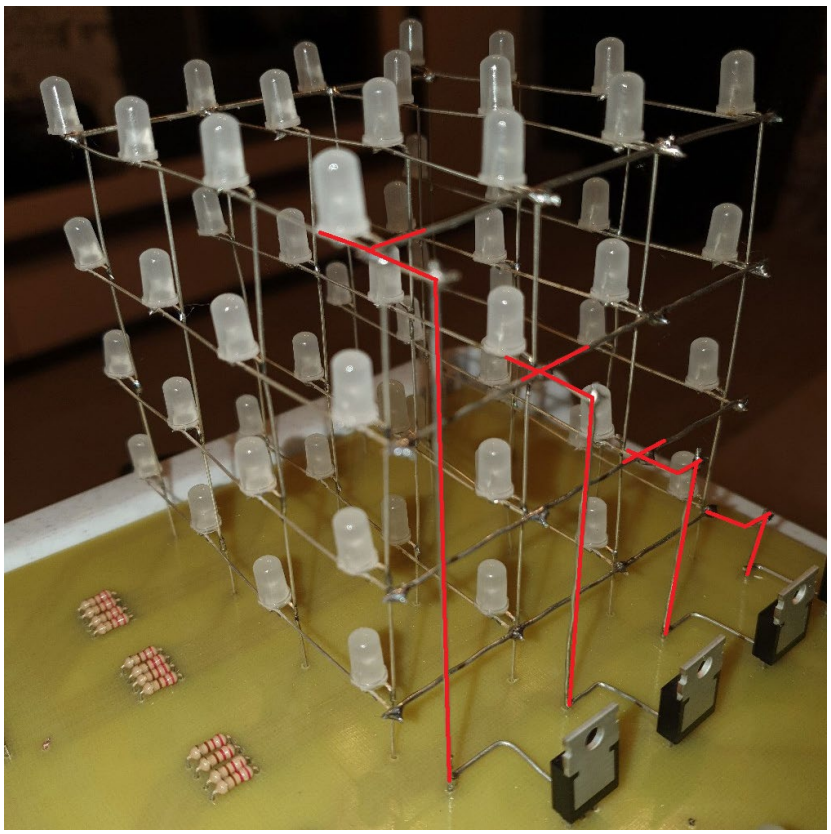


Рисунок 8 – Подключение слоев светодиодного куба к коллекторам биполярных транзисторов

После пайки компонентов на плату, без подключения LCD дисплея и тумблера, результат показан на рисунках 9 и 10.

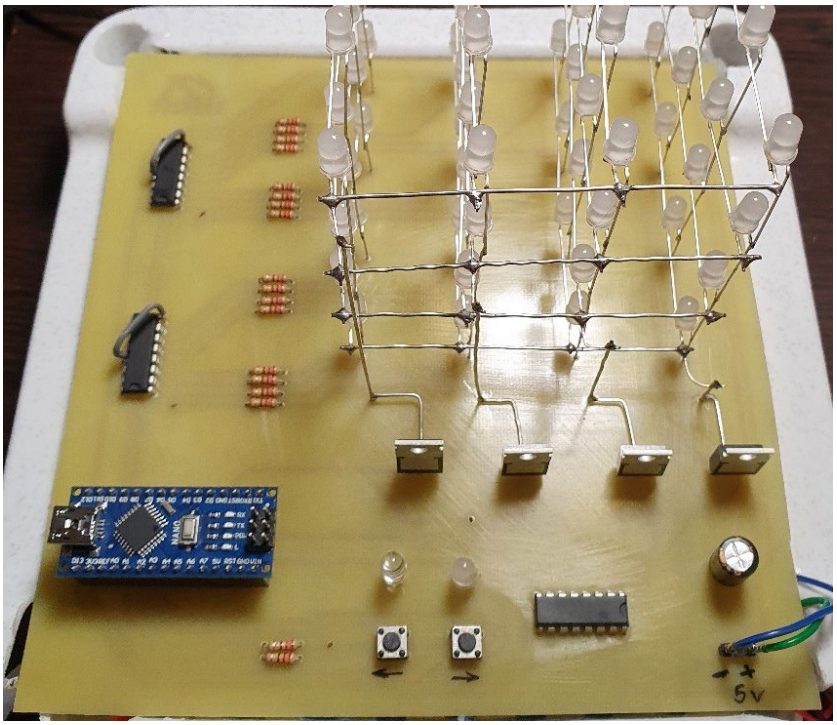


Рисунок 9 – Плата управления в сборке. Вид сверху

Рисунок 10 – Плата управления в сборке. Вид снизу

Дополнительно, было принято решение поместить всю конструкцию в корпус, на котором располагается переключатель для подачи питания к плате, а также LCD дисплей, который подключен по I2C к аналоговым разъемам A4 и A5 на плате Arduino Nano (SDA подключается к A4, а SCL к A5). Вид конструкции спереди приведен на рисунке 11.

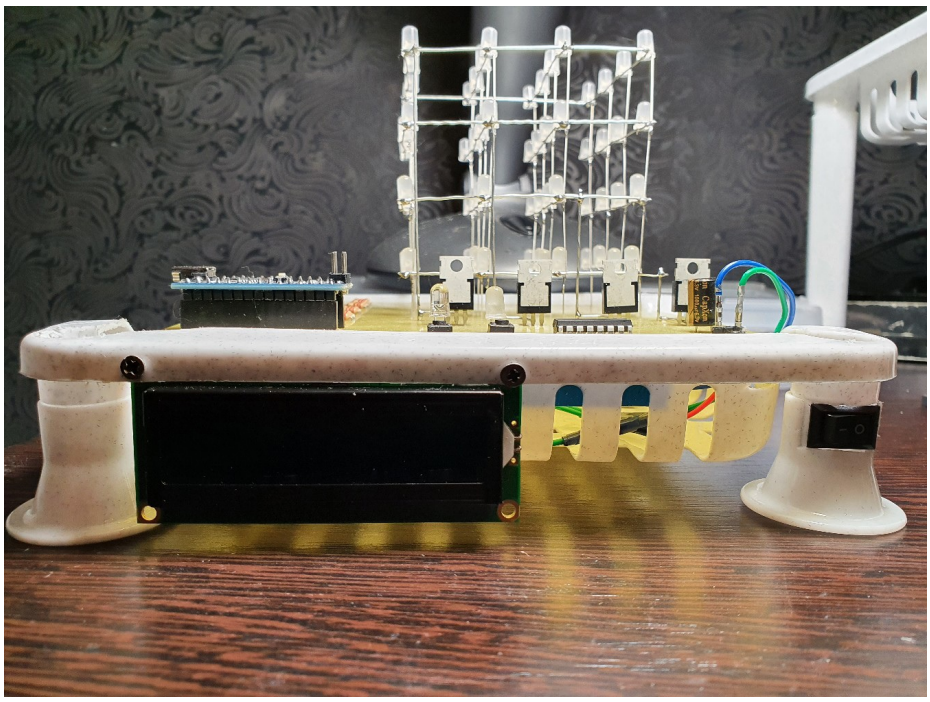


Рисунок 11 – Плата управления светодиодным кубом. Вид спереди

Плата питается от блока питания на 5 вольт и 2 ампера, подключенного в сеть 220В. После переключения тумблера, питание идет на Arduino Nano, и светодиодный куб начинает работать. Результат сборки конструкции приведен на рисунке 12.

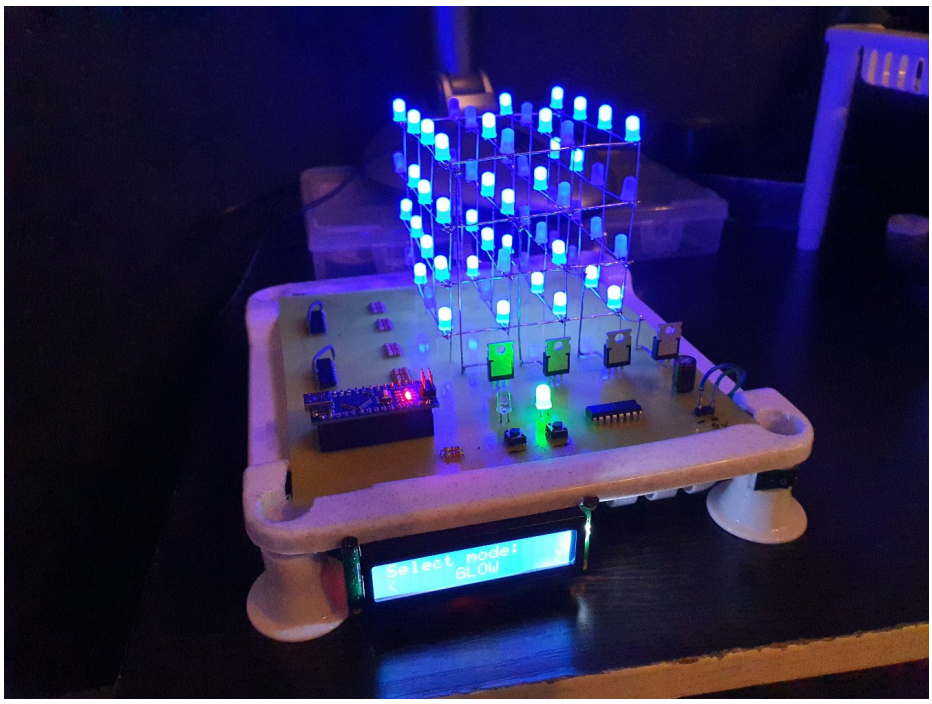


Рисунок 12 – результат сборки платы управления светодиодным кубом 4x4x4

Результаты

Подводя итоги, следует обобщить, что было сделано в ходе курсового проекта. Мною был выполнен синтез принципиальной схемы, разводка и изготовление печатной платы методом ЛУТ, монтаж радиокомпонентов на плате. Также была разработана программа в среде Arduino IDE для прошивки контроллера Arduino Nano. В результате получился стендовой вариант светодиодного куба с различными режимами подсветки. В заключение хочется добавить ссылку на проект на сервисе GitHub со схемами, пояснениями и выполненной разводкой платы. [11]

Литература

1 Сдвиговые регистры [Электронный ресурс] // 3D-DIY.RU : База знаний. URL : https://3d-diy.ru/wiki/components/sdvigovye-registry (дата обращения: 05.05.2021)

2 Платформа Arduino [Электронный ресурс] // ARDUINO.CC : Официальный сайт компании производителя. URL : https://www.arduino.cc (дата обращения: 05.05.2021)

3 Семейство микроконтроллеров AVR со сверхнизким энергопотреблением picoPower [Электронный ресурс] // KIT-E.RU : Компоненты и технологии. URL : https://kit-e.ru/micro/semejstvo-mikrokontrollerov-avr-so-sverhnizkim-energopotrebleniem-picopower (дата обращения: 10.05.2021)

4 Сравнение микрокотнтроллеров Atmega328P и Atmega168 [Электронный ресурс] // FINDCHIPS.COM : Техническая документация. URL : https://www.findchips.com/compare/ATMEGA328P-MUR--vs--ATMEGA168-20PU (дата обращения: 12.05.2021)

5 Measuring Memory Usage [Электронный ресурс] // URL : https://learn.adafruit.com/memories-of-an-arduino/measuring-free-memory (дата обращения: 08.05.2021)

6 Динамическая индикация. Программирование микроконтроллеров AVR [Электронный ресурс] // DIODOV.NET : Школа электроники. URL : https://diodov.net/dinamicheskaya-indikatsiya-programmirovanie-mikrokontrollerov-avr (дата обращения: 06.05.2021)

7 Среда разработки EasyEDA [Электронный ресурс] // URL : https://easyeda.com/

8 Последовательный интерфейс SPI [Электронный ресурс] // GAV.RU : Справочник по электронным компанентам. URL : http://www.gaw.ru/ html.cgi/txt/interface/spi/index.htm (дата обращения: 06.05.2021)

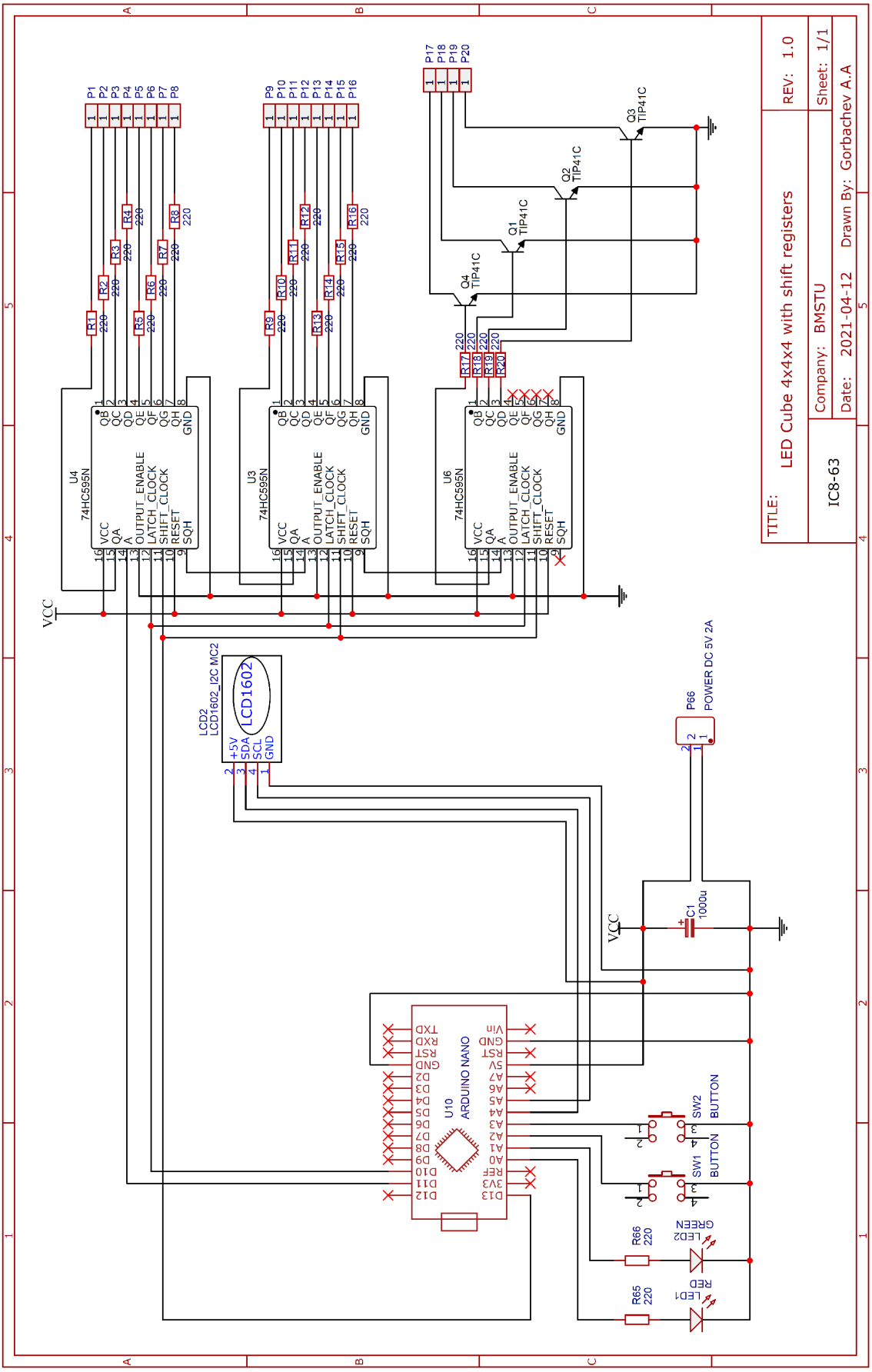
9 Подключение дисплея LCD 1602 [Электронный ресурс] // ARDUINOMASTER.RU : Справочник по Arduino. URL : https://arduinomaster.ru/datchiki-arduino/lcd-i2c-arduino-displey-ekran (дата обращения: 06.05.2021)

10 Изготовление печатных плат по ЛУТ [Электронный ресурс] // статья. URL : https://habr.com/ru/post/451314 (дата обращения: 20.05.2021)

11 Проект светодиодного куба [Электронный ресурс] // URL : https://github.com/OzoNeTT/eis-coursework

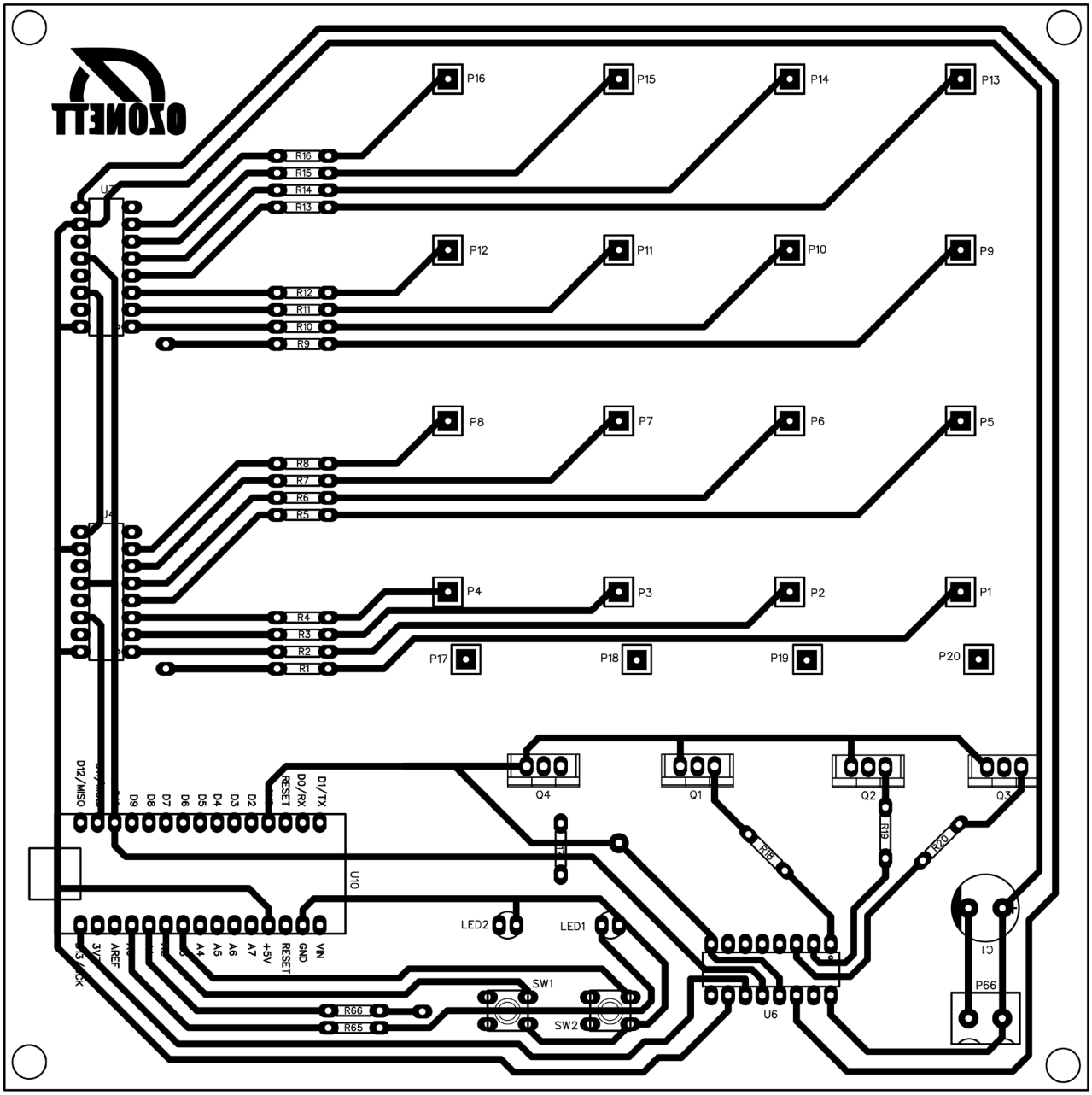
Приложение А

Принципиальная схема



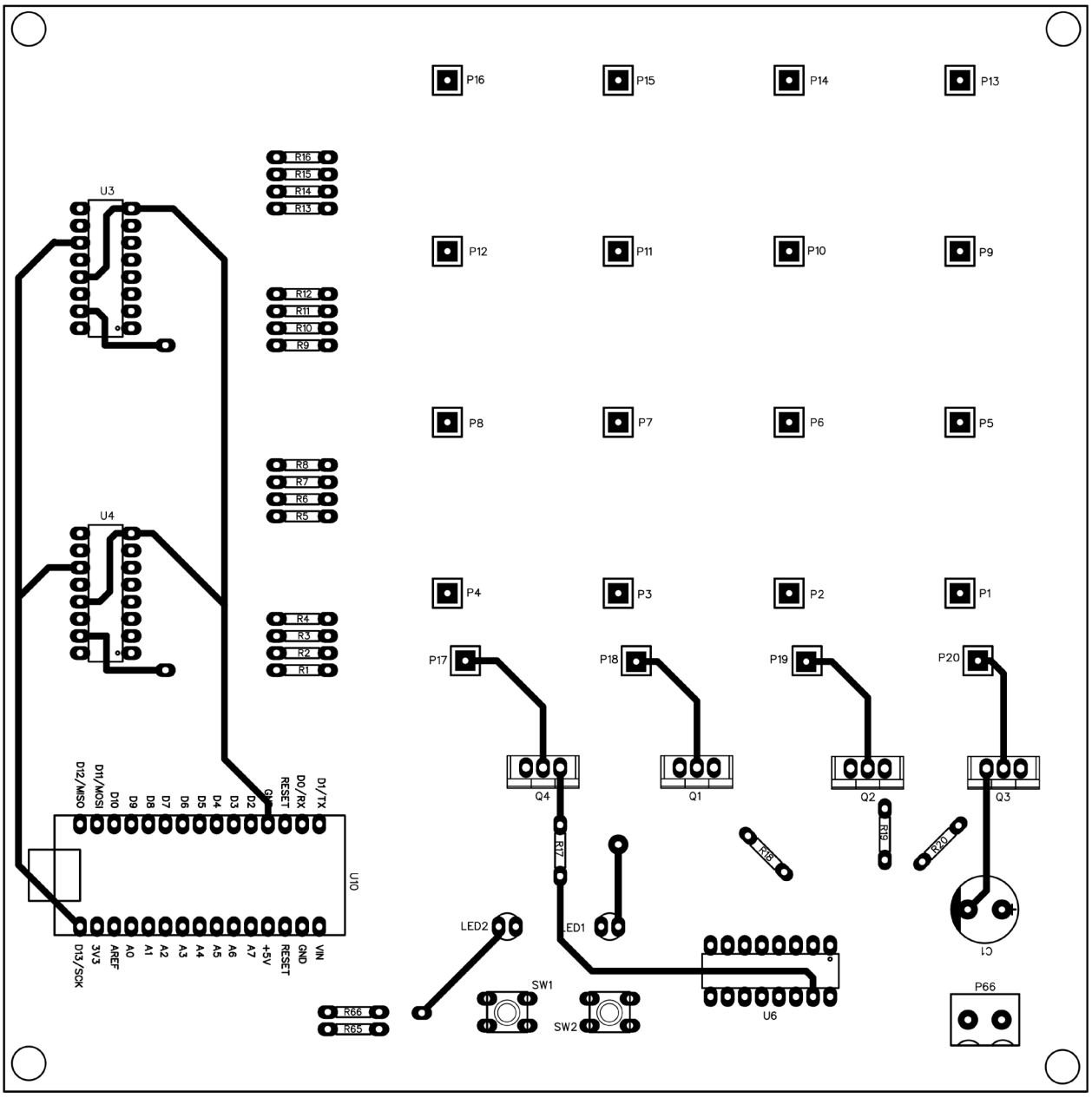
Приложение Б

Печатная плата нижний слой



Приложение В

Печатная плата верхний слой



Приложение Г

Исходный код программы для прошивки Arduino Nano

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#define INVERT_Y 1    // инвертировать по вертикали (если дождь идёт вверх)
#define INVERT_X 0    // инвертировать по горизонтали (если текст не читается)
#define XAXIS 0
#define YAXIS 1
#define ZAXIS 2
#define POS_X 0
#define NEG_X 1
#define POS_Z 2
#define NEG_Z 3
#define POS_Y 4
#define NEG_Y 5
#define Up_buttonPin A3
#define Down_buttonPin A2
#define RED_LED A1
#define GREEN_LED A0
//control buttons -----------
int buttonPushCounter = 0;   // counter for the number of button presses
int up_buttonState = 0;         // current state of the up button
int up_lastButtonState = 0;     // previous state of the up button
int down_buttonState = 0;         // current state of the up button
int down_lastButtonState = 0;     // previous state of the up button
bool bPress = false;
// ---------------------------
// effects duration ----------
#define RAIN_TIME 260
#define PLANE_BOING_TIME 220
#define SEND_VOXELS_TIME 140
#define WOOP_WOOP_TIME 350
#define CUBE_JUMP_TIME 200
#define CLOCK_TIME 500
#define GLOW_TIME 8
#define WALKING_TIME 100
// ---------------------------
int currentEffect; // for glowing
uint8_t cube[8][8];
uint32 t millTimer;
uint16 t timer;
uint16_t modeTimer;
bool loading;
int8 t pos;
int8_t vector[3];
int16 t coord[3];
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
void setup()
{
    Serial.begin(9600);
    lcd.begin(16, 2);
    lcd.setCursor(0, 0);
    lcd.print("Select mode:");
    SPI.begin();
    SPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE0));
    randomSeed(analogRead(0));
    pinMode(Up_buttonPin,INPUT_PULLUP);
    pinMode(Down_buttonPin,INPUT PULLUP);
    pinMode(RED_LED, OUTPUT);
    pinMode(GREEN_LED, OUTPUT);
    lcd.setCursor(0,1);
    lcd.print("<     GLOW     >");

    loading = true;
    currentEffect = 0;
    changeMode();

}
void loop()
{
    checkUp();
    checkDown();
    if( bPress){
        bPress = false;
        currentEffect = abs(buttonPushCounter) % 10;
        changeMode();
```

```
    }
    switch (currentEffect) {
        case 1: rain(); break;
        case 2: planeBoing(); break;
        case 3: sendVoxels(); break;
        case 4: woopWoop(); break;
        case 5: cubeJump(); break;
        case 6: lit(); break;
        case 7: sinusFill(); break;
        case 8: sinusThin(); break;
        case 9: walkingCube(); break;
        case 0: glow(); break;
    }
    renderCube();
}

byte func1(byte x){
    return ((x&1) | ((x >> 2) << 1 )) & 3;
}

byte func2(byte x){
    return (((x & 128) | ((x << 2) >> 1)) & (128 + 64)) >> 4;
}

void renderCube() {
    digitalWrite(SS, LOW);
    SPI.transfer(0x01 << 0);

    byte num11 = func2(cube[0][2] & (~31)) | func1(cube[0][2] & 31)  |  (func2(cube[0][0] & (~31)) |
func1(cube[0][0] & 31)) << 4;
    SPI.transfer(num11);
    byte num12 = func2(cube[0][7] & (~31)) | func1(cube[0][7] & 31)  |  (func2(cube[0][5] & (~31)) |
func1(cube[0][5] & 31)) << 4;
    SPI.transfer(num12);
    digitalWrite (SS, HIGH);
    delayMicroseconds(50);

    digitalWrite (SS, LOW);
    SPI.transfer(0x01 << 1);

    byte num21 = func2(cube[2][2] & (~31)) | func1(cube[2][2] & 31)  |  (func2(cube[2][0] & (~31)) |
func1(cube[2][0] & 31)) << 4;
    SPI.transfer(num21);
    byte num22 = func2(cube[2][7] & (~31)) | func1(cube[2][7] & 31)  |  (func2(cube[2][5] & (~31)) |
func1(cube[2][5] & 31)) << 4;
    SPI.transfer(num22);
    digitalWrite (SS, HIGH);
    delayMicroseconds(50);

    digitalWrite (SS, LOW);
    SPI.transfer(0x01 << 2);

    byte num31 = func2(cube[5][2] & (~31)) | func1(cube[5][2] & 31)  |  (func2(cube[5][0] & (~31)) |
func1(cube[5][0] & 31)) << 4;
    SPI.transfer(num31);
    byte num32 = func2(cube[5][7] & (~31)) | func1(cube[5][7] & 31)  |  (func2(cube[5][5] & (~31)) |
func1(cube[5][5] & 31)) << 4;
    SPI.transfer(num32);
    digitalWrite (SS, HIGH);
    delayMicroseconds(50);

    digitalWrite (SS, LOW);
    SPI.transfer(0x01 << 3);

    byte num41 = func2(cube[7][2] & (~31)) | func1(cube[7][2] & 31)  |  (func2(cube[7][0] & (~31)) |
func1(cube[7][0] & 31)) << 4;
    SPI.transfer(num41);
    byte num42 = func2(cube[7][7] & (~31)) | func1(cube[7][7] & 31)  |  (func2(cube[7][5] & (~31)) |
func1(cube[7][5] & 31)) << 4;
    SPI.transfer(num42);
    digitalWrite (SS, HIGH);
    delayMicroseconds(50);
}

void changeMode() {
    clearCube();
    loading = true;
    timer = 0;
    randomSeed(millis());
    digitalWrite(RED_LED, HIGH);
```

```
    digitalWrite(GREEN_LED, LOW);
    delay(500);
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, HIGH);

    switch (currentEffect) {
        case 1:
            lcd.setCursor(0,1);
            lcd.print("<     RAIN     >");
            modeTimer = RAIN_TIME;
            break;
        case 2:
            lcd.setCursor(0,1);
            lcd.print("<    PLANE     >");
            modeTimer = PLANE_BOING_TIME;
            break;
        case 3:
            lcd.setCursor(0,1);
            lcd.print("<    VOXELS    >");
            modeTimer = SEND_VOXELS_TIME;
            break;
        case 4:
            lcd.setCursor(0,1);
            lcd.print("<     WOOP     >");
            modeTimer = WOOP_WOOP_TIME;
            break;
        case 5:
            lcd.setCursor(0,1);
            lcd.print("<     JUMP     >");
            modeTimer = CUBE_JUMP_TIME;
            break;
        case 6:
            lcd.setCursor(0,1);
            lcd.print("<      LIT     >");
            modeTimer = CLOCK_TIME;
            break;
        case 7:
            lcd.setCursor(0,1);
            lcd.print("<     SIN 1    >");
            modeTimer = RAIN_TIME;
            break;
        case 8:
            lcd.setCursor(0,1);
            lcd.print("<     SIN 2    >");
            modeTimer = RAIN_TIME;
            break;
        case 9:
            lcd.setCursor(0,1);
            lcd.print("<    WALKING   >");
            modeTimer = WALKING_TIME;
            break;
        case 0:
            lcd.setCursor(0,1);
            lcd.print("<     GLOW     >");
            modeTimer = GLOW_TIME;
            break;

    }
}

void checkUp()
{
    up_buttonState = digitalRead(Up_buttonPin);
    if (up buttonState != up lastButtonState) {
        if (up buttonState == LOW) {
            bPress = true;
            buttonPushCounter++;
        }
        delay(50);
    }
    up lastButtonState = up buttonState;
}

void checkDown()
{
    down_buttonState = digitalRead(Down_buttonPin);
    if (down buttonState != down lastButtonState) {
        if (down_buttonState == LOW) {
            bPress = true;
            if (buttonPushCounter - 1 < 0){
```

```
                buttonPushCounter = 9;
            } else {
                buttonPushCounter--;
            }
        } else {

        }
        delay(50);
    }
    down_lastButtonState = down_buttonState;

}

void walkingCube() {
    if (loading) {
        clearCube();
        loading = false;
        for (byte i = 0; i < 3; i++) {
            coord[i] = 300;
            vector[i] = random(3, 8) * 15;
        }
    }
    timer++;
    if (timer > modeTimer) {
        timer = 0;
        clearCube();
        for (byte i = 0; i < 3; i++) {
            coord[i] += vector[i];
            if (coord[i] < 1) {
                coord[i] = 1;
                vector[i] = -vector[i];
                vector[i] += random(0, 6) - 3;
            }
            if (coord[i] > 700 - 100) {
                coord[i] = 700 - 100;
                vector[i] = -vector[i];
                vector[i] += random(0, 6) - 3;
            }
        }
        int8_t thisX = coord[0] / 100;
        int8_t thisY = coord[1] / 100;
        int8_t thisZ = coord[2] / 100;

        setVoxel(thisX, thisY, thisZ);
        setVoxel(thisX + 1, thisY, thisZ);
        setVoxel(thisX, thisY + 1, thisZ);
        setVoxel(thisX, thisY, thisZ + 1);
        setVoxel(thisX + 1, thisY + 1, thisZ);
        setVoxel(thisX, thisY + 1, thisZ + 1);
        setVoxel(thisX + 1, thisY, thisZ + 1);
        setVoxel(thisX + 1, thisY + 1, thisZ + 1);
    }
}

void sinusFill() {
    if (loading) {
        clearCube();
        loading = false;
    }
    timer++;
    if (timer > modeTimer) {
        timer = 0;
        clearCube();
        if (++pos > 10) pos = 0;
        for (uint8_t i = 0; i < 8; i++) {
            for (uint8_t j = 0; j < 8; j++) {
                int8_t sinZ = 4 + ((float)sin((float)(i + pos) / 2) * 2);
                for (uint8_t y = 0; y < sinZ; y++) {
                    setVoxel(i, y, j);
                }
            }
        }
    }
}

void sinusThin() {
    if (loading) {
        clearCube();
        loading = false;
    }
```

```
    timer++;
    if (timer > modeTimer) {
        timer = 0;
        clearCube();
        if (++pos > 10) pos = 0;
        for (uint8_t i = 0; i < 8; i++) {
            for (uint8_t j = 0; j < 8; j++) {
                int8 t sinZ = 4 + ((float)sin((float)(i + pos) / 2) * 2);
                setVoxel(i, sinZ, j);
            }
        }
    }
}

void rain() {
    if (loading) {
        clearCube();
        loading = false;
    }
    timer++;
    if (timer > modeTimer) {
        timer = 0;
        shift(NEG_Y);
        uint8_t numDrops = random(0, 5);
        for (uint8 t i = 0; i < numDrops; i++) {
            setVoxel(random(0, 8), 7, random(0, 8));
        }
    }
}

uint8 t planePosition = 0;
uint8_t planeDirection = 0;
bool looped = false;

void planeBoing() {
    if (loading) {
        clearCube();
        uint8_t axis = random(0, 3);
        planePosition = random(0, 2) * 7;
        setPlane(axis, planePosition);
        if (axis == XAXIS) {
            if (planePosition == 0) {
                planeDirection = POS_X;
            } else {
                planeDirection = NEG_X;
            }
        } else if (axis == YAXIS) {
            if (planePosition == 0) {
                planeDirection = POS_Y;
            } else {
                planeDirection = NEG_Y;
            }
        } else if (axis == ZAXIS) {
            if (planePosition == 0) {
                planeDirection = POS_Z;
            } else {
                planeDirection = NEG_Z;
            }
        }
        timer = 0;
        looped = false;
        loading = false;
    }

    timer++;
    if (timer > modeTimer) {
        timer = 0;
        shift(planeDirection);
        if (planeDirection % 2 == 0) {
            planePosition++;
            if (planePosition == 7) {
                if (looped) {
                    loading = true;
                } else {
                    planeDirection++;
                    looped = true;
                }
            }
        } else {
            planePosition--;
```

```
            if (planePosition == 0) {
                if (looped) {
                    loading = true;
                } else {
                    planeDirection--;
                    looped = true;
                }
            }
        }
    }
}

uint8_t selX = 0;
uint8_t selY = 0;
uint8_t selZ = 0;
uint8_t sendDirection = 0;
bool sending = false;

void sendVoxels() {
    if (loading) {
        clearCube();
        for (uint8_t x = 0; x < 8; x++) {
            for (uint8_t z = 0; z < 8; z++) {
                setVoxel(x, random(0, 2) * 7, z);
            }
        }
        loading = false;
    }

    timer++;
    if (timer > modeTimer) {
        timer = 0;
        if (!sending) {
            selX = random(0, 8);
            selZ = random(0, 8);
            if (getVoxel(selX, 0, selZ)) {
                selY = 0;
                sendDirection = POS_Y;
            } else if (getVoxel(selX, 7, selZ)) {
                selY = 7;
                sendDirection = NEG_Y;
            }
            sending = true;
        } else {
            if (sendDirection == POS_Y) {
                selY++;
                setVoxel(selX, selY, selZ);
                clearVoxel(selX, selY - 1, selZ);
                if (selY == 7) {
                    sending = false;
                }
            } else {
                selY--;
                setVoxel(selX, selY, selZ);
                clearVoxel(selX, selY + 1, selZ);
                if (selY == 0) {
                    sending = false;
                }
            }
        }
    }
}

uint8_t cubeSize = 0;
bool cubeExpanding = true;

void woopWoop() {
    if (loading) {
        clearCube();
        cubeSize = 2;
        cubeExpanding = true;
        loading = false;
    }

    timer++;
    if (timer > modeTimer) {
        timer = 0;
        if (cubeExpanding) {
            cubeSize += 2;
            if (cubeSize == 8) {
```

```
                cubeExpanding = false;
            }
        } else {
            cubeSize -= 2;
            if (cubeSize == 2) {
                cubeExpanding = true;
            }
        }
        clearCube();
        drawCube(4 - cubeSize / 2, 4 - cubeSize / 2, 4 - cubeSize / 2, cubeSize);
    }
}

uint8 t xPos;
uint8 t yPos;
uint8_t zPos;

void cubeJump() {
    if (loading) {
        clearCube();
        xPos = random(0, 2) * 7;
        yPos = random(0, 2) * 7;
        zPos = random(0, 2) * 7;
        cubeSize = 8;
        cubeExpanding = false;
        loading = false;
    }

    timer++;
    if (timer > modeTimer) {
        timer = 0;
        clearCube();
        if (xPos == 0 && yPos == 0 && zPos == 0) {
            drawCube(xPos, yPos, zPos, cubeSize);
        } else if (xPos == 7 && yPos == 7 && zPos == 7) {
            drawCube(xPos + 1 - cubeSize, yPos + 1 - cubeSize, zPos + 1 - cubeSize, cubeSize);
        } else if (xPos == 7 && yPos == 0 && zPos == 0) {
            drawCube(xPos + 1 - cubeSize, yPos, zPos, cubeSize);
        } else if (xPos == 0 && yPos == 7 && zPos == 0) {
            drawCube(xPos, yPos + 1 - cubeSize, zPos, cubeSize);
        } else if (xPos == 0 && yPos == 0 && zPos == 7) {
            drawCube(xPos, yPos, zPos + 1 - cubeSize, cubeSize);
        } else if (xPos == 7 && yPos == 7 && zPos == 0) {
            drawCube(xPos + 1 - cubeSize, yPos + 1 - cubeSize, zPos, cubeSize);
        } else if (xPos == 0 && yPos == 7 && zPos == 7) {
            drawCube(xPos, yPos + 1 - cubeSize, zPos + 1 - cubeSize, cubeSize);
        } else if (xPos == 7 && yPos == 0 && zPos == 7) {
            drawCube(xPos + 1 - cubeSize, yPos, zPos + 1 - cubeSize, cubeSize);
        }
        if (cubeExpanding) {
            cubeSize++;
            if (cubeSize == 8) {
                cubeExpanding = false;
                xPos = random(0, 2) * 7;
                yPos = random(0, 2) * 7;
                zPos = random(0, 2) * 7;
            }
        } else {
            cubeSize--;
            if (cubeSize == 1) {
                cubeExpanding = true;
            }
        }
    }
}

bool glowing;
uint16 t glowCount = 0;

void glow() {
    if (loading) {
        clearCube();
        glowCount = 0;
        glowing = true;
        loading = false;
    }

    timer++;
    if (timer > modeTimer) {
        timer = 0;
```

```
        if (glowing) {
            if (glowCount < 448) {
                do {
                    selX = random(0, 8);
                    selY = random(0, 8);
                    selZ = random(0, 8);
                } while (getVoxel(selX, selY, selZ));
                setVoxel(selX, selY, selZ);
                glowCount++;
            } else if (glowCount < 512) {
                lightCube();
                glowCount++;
            } else {
                glowing = false;
                glowCount = 0;
            }
        } else {
            if (glowCount < 448) {
                do {
                    selX = random(0, 8);
                    selY = random(0, 8);
                    selZ = random(0, 8);
                } while (!getVoxel(selX, selY, selZ));
                clearVoxel(selX, selY, selZ);
                glowCount++;
            } else {
                clearCube();
                glowing = true;
                glowCount = 0;
            }
        }
    }
}


void lit() {
    if (loading) {
        clearCube();
        for (uint8_t i = 0; i < 8; i++) {
            for (uint8_t j = 0; j < 8; j++) {
                cube[i][j] = 0xFF;
            }
        }
        loading = false;
    }
}

void setVoxel(uint8_t x, uint8_t y, uint8_t z) {
    cube[7 - y][7 - z] |= (0x01 << x);
}

void clearVoxel(uint8_t x, uint8_t y, uint8_t z) {
    cube[7 - y][7 - z] ^= (0x01 << x);
}

bool getVoxel(uint8_t x, uint8_t y, uint8_t z) {
    return (cube[7 - y][7 - z] & (0x01 << x)) == (0x01 << x);
}

void setPlane(uint8 t axis, uint8 t i) {
    for (uint8_t j = 0; j < 8; j++) {
        for (uint8_t k = 0; k < 8; k++) {
            if (axis == XAXIS) {
                setVoxel(i, j, k);
            } else if (axis == YAXIS) {
                setVoxel(j, i, k);
            } else if (axis == ZAXIS) {
                setVoxel(j, k, i);
            }
        }
    }
}

void shift(uint8_t dir) {

    if (dir == POS_X) {
        for (uint8 t y = 0; y < 8; y++) {
            for (uint8_t z = 0; z < 8; z++) {
                cube[y][z] = cube[y][z] << 1;
            }
```

```
        }
    } else if (dir == NEG_X) {
        for (uint8_t y = 0; y < 8; y++) {
            for (uint8_t z = 0; z < 8; z++) {
                cube[y][z] = cube[y][z] >> 1;
            }
        }
    } else if (dir == POS_Y) {
        for (uint8_t y = 1; y < 8; y++) {
            for (uint8_t z = 0; z < 8; z++) {
                cube[y - 1][z] = cube[y][z];
            }
        }
        for (uint8_t i = 0; i < 8; i++) {
            cube[7][i] = 0;
        }
    } else if (dir == NEG_Y) {
        for (uint8_t y = 7; y > 0; y--) {
            for (uint8_t z = 0; z < 8; z++) {
                cube[y][z] = cube[y - 1][z];
            }
        }
        for (uint8_t i = 0; i < 8; i++) {
            cube[0][i] = 0;
        }
    } else if (dir == POS_Z) {
        for (uint8_t y = 0; y < 8; y++) {
            for (uint8_t z = 1; z < 8; z++) {
                cube[y][z - 1] = cube[y][z];
            }
        }
        for (uint8_t i = 0; i < 8; i++) {
            cube[i][7] = 0;
        }
    } else if (dir == NEG_Z) {
        for (uint8_t y = 0; y < 8; y++) {
            for (uint8_t z = 7; z > 0; z--) {
                cube[y][z] = cube[y][z - 1];
            }
        }
        for (uint8_t i = 0; i < 8; i++) {
            cube[i][0] = 0;
        }
    }
}

void drawCube(uint8_t x, uint8_t y, uint8_t z, uint8_t s) {
    for (uint8_t i = 0; i < s; i++) {
        setVoxel(x, y + i, z);
        setVoxel(x + i, y, z);
        setVoxel(x, y, z + i);
        setVoxel(x + s - 1, y + i, z + s - 1);
        setVoxel(x + i, y + s - 1, z + s - 1);
        setVoxel(x + s - 1, y + s - 1, z + i);
        setVoxel(x + s - 1, y + i, z);
        setVoxel(x, y + i, z + s - 1);
        setVoxel(x + i, y + s - 1, z);
        setVoxel(x + i, y, z + s - 1);
        setVoxel(x + s - 1, y, z + i);
        setVoxel(x, y + s - 1, z + i);
    }
}

void lightCube() {
    for (uint8_t i = 0; i < 8; i++) {
        for (uint8_t j = 0; j < 8; j++) {
            cube[i][j] = 0xFF;
        }
    }
}

void clearCube() {
    for (uint8_t i = 0; i < 8; i++) {
        for (uint8_t j = 0; j < 8; j++) {
            cube[i][j] = 0;
        }
    }
}
```