министерство науки и Высшего обРазования РОССИЙСКОЙ федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тихоокеанский государственный университет»

|  |  |
| --- | --- |
| Институт | Политехнический |
| Высшая школа | Кибернетики и цифровых технологий |
| Направление | 09.03.01 Информатика и вычислительная техника |
|  | (шифр, наименование) |
| Профиль | Компьютерные сети и телекоммуникации |

ДОПУСТИТЬ К ЗАЩИТЕ

Руководитель ВШ КЦТ \_\_\_\_\_\_\_\_\_ Давыдов О.А.

подпись ФИО

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Тема Разработка микропроцессорной системы для строительного 3D-принтера \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_\_\_\_\_\_\_\_ Шушаков А.А.

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_ Зинкевич А. В.

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

Нормоконтролёр \_\_\_\_\_\_\_\_\_\_\_ Зинкевич А. В.

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

**Консультанты:**

По \_\_\_\_\_\_\_\_\_\_\_ \_ \_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

дата

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_

дата

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_

Дата

**Хабаровск – 2025 г.**

министерство науки и Высшего обРазования РОССИЙСКОЙ федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тихоокеанский государственный университет»

|  |  |
| --- | --- |
| Институт | Политехнический |
| Высшая школа | Кибернетики и цифровых технологий |
| Направление | 09.03.01 Информатика и вычислительная техника |
|  | (шифр, наименование) |
| Профиль | Компьютерные сети и телекоммуникации |

ДОПУСТИТЬ К ЗАЩИТЕ

Руководитель ВШ КЦТ \_\_\_\_\_\_\_\_\_ Давыдов О.А.

подпись ФИО

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**З А Д А Н И Е**

**на выпускную квалификационную работу**

Студенту Шушакову Андрею Андреевичу

1. Тема работы \_\_\_ Разработка микропроцессорной системы для строительного 3D-принтера \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утверждена приказом по университету № \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

2. Срок сдачи студентом выпускной квалификационной работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

3. Исходные данные к работе Техническая документация на микроконтроллеры Arduino, асинхронные двигатели ЭТФ-500, шаговые двигатели HB86HS114-4600А и драйвера CW8060 к ним; программное обеспечение Arduino IDE для разработки кода; Справочная литература по строительной 3D-печати.

4. Перечень подлежащих разработке в выпускной квалификационной работе вопросов:

1. Проектирование архитектуры системы управления строительной 3D-печатью на базе микроконтроллеров Arduino;

2. Синхронизация асинхронных двигателей ЭТФ-500 по оси Z.

3. Написание программного обеспечения по обработке G-кода для микроконтроллера Arduino

5. Перечень графического материала (с точным указание обязательных чертежей), либо раздаточного материала К**омпоненты системы управления 3D-принтером, электрические схемы и обозначения, листинги программ.**

6. Консультанты:

|  |  |
| --- | --- |
| Раздел работы | ФИО, подпись, дата |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Руководитель работы Задание принял к исполнению студент

\_\_\_\_\_\_\_\_\_\_\_\_ Зинкевич А. В. \_\_\_\_\_\_\_\_\_\_\_\_ Шушаков А.А.

подпись ФИО подпись ФИО

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

дата дата

РЕФЕРАТ

Выпускная квалификационная работа содержит 61 страниц текстового документа формата А4, 13 рисунков, 4 таблицы, 15 источников, 2 приложения.

Объект разработки: строительный 3D-принтер.

Цель работы: разработка системы автоматизированной системы управления строительным 3D-принтером.

Методология разработки: системный анализ, программирование микроконтроллеров, тестирование. Использованы шаговые двигатели, однофазные двигатели, трёхфазный асинхронный двигатель, микроконтроллеры Arduino, датчики Холла, индуктивные концевые датчики, драйверы шаговых двигателей, частотный преобразователь.

Разработана система, обеспечивающая, синхронизацию четырех асинхронных двигателей по оси Z, управление шаговыми двигателями по осям X и Y, позиционирование с помощью концевых датчиков, подачу бетона, обработку аварийных ситуаций, чтение G-кода с карты памяти и его обработку в микроконтроллере. Для помехоустойчивой связи микроконтроллеров на расстоянии до 10 метров применён протокол передачи данных Modbus RTU.

Технико-эксплуатационные характеристики: система обеспечивает точность позиционирования печатающей головки до 0,5 миллиметров по осям X и Y и до 10 миллиметров по оси Z, устойчивость к сбоям связи.

Область применения: производство бетонных изделий в строительной отрасли, образовательные учреждения для изучения технологий 3D-печати.

Ключевые слова: СТРОИТЕЛЬНАЯ 3D-ПЕЧАТЬ, МИКРОКОНТРОЛЛЕР ARDUINO, АСИНХРОННЫЕ ДВИГАТЕЛИ, ШАГОВЫЕ ДВИГАТЕЛИ, ДАТЧИКИ ХОЛЛА, КОНЦЕВЫЕ ДАТЧИКИ, G-КОД, MODBUS RTU, ПОДАЧА БЕТОНА, ПОМЕХОУСТОЙЧИВАЯ СВЯЗЬ, АВАРИЙНОЕ СОСТОЯНИЕ.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc201599283)

[ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ АВТОМАТИЗИРОВАННОГО УПРАВЛЕНИЯ СТРОИТЕЛЬНЫМ 3D-ПРИНТЕРОМ 10](#_Toc201599284)

[1.1 Виды строительных 3D-принтеров 10](#_Toc201599285)

[1.2 Компоненты платформенных строительных 3D-принтеров 12](#_Toc201599286)

[1.3 G-код в 3D-печати. 13](#_Toc201599287)

[ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ 16](#_Toc201599288)

[2.1 Принципы синхронизации однофазных асинхронных двигателей 16](#_Toc201599289)

[2.2 Передача данных по протоколу Modbus RTU на общей шине RS485. 17](#_Toc201599290)

[2.3 Проектирование системы 20](#_Toc201599291)

[ГЛАВА 3. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ 23](#_Toc201599292)

[3.1 Аппаратная реализация 23](#_Toc201599293)

[3.2 Программная реализация 27](#_Toc201599294)

[3.3 Тестирование системы 32](#_Toc201599295)

[ЗАКЛЮЧЕНИЕ 34](#_Toc201599296)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 37](#_Toc201599297)

[ПРИЛОЖЕНИЕ А 39](#_Toc201599298)

[ПРИЛОЖЕНИЕ Б 40](#_Toc201599299)

# ВВЕДЕНИЕ

Актуальность темы обусловлена возрастающим интересом к строительной 3D-печати, которая, несмотря на ограниченное распространение, демонстрирует значительный потенциал в последние годы [1]. Технология особенно перспективна в малоэтажном строительстве и производстве малогабаритных изделий, таких как элементы декора и мебель, благодаря высокой скорости, низкой стоимости и стабильному качеству по сравнению с традиционными методами. Однако успешное применение 3D-печати требует разработки надёжных систем управления, обеспечивающих точное позиционирование печатающей головки и синхронизацию всех компонентов. Создание доступной системы управления на базе микроконтроллера Arduino Uno, способной эффективно обрабатывать G-код и учитывать аварийные ситуации, представляет собой актуальную задачу, способствующую популяризации строительной 3D-печати.

Целью работы является разработка системы управления строительным 3D-принтером для автоматизированной печати малогабаритных бетонных изделий с использованием микроконтроллера Arduino Uno.

Задачи исследования:

* Провести анализ существующих систем управления строительными 3D-принтерами и их программно-аппаратных решений.
* Разработать алгоритмы обработки G-кода с SD-карты для управления движением печатающей головки по осям X, Y, Z.
* Реализовать синхронизацию асинхронных двигателей по оси Z с использованием датчиков Холла и концевых датчиков для хоминга.
* Обеспечить управление подачей бетона и обработку аварийных ситуаций.
* Провести тестирование и отладку системы на физическом прототипе.

Объектом исследования является процесс управления строительным 3D-принтером для печати малогабаритных бетонных изделий. Предметом исследования выступает программно-аппаратная система управления на базе микроконтроллера Arduino Uno, включающая обработку G-кода, синхронизацию двигателей и управление подачей материала.

Методологической основой исследования являются принципы системного анализа, методы проектирования встраиваемых систем и подходы к обработке G-кода в системах числового программного управления. Теоретическая база включает изучение принципов работы строительных 3D-принтеров, асинхронных и шаговых двигателей, датчиков Холла, концевых датчиков и протокола Modbus RTU. В ходе работы проанализированы научные публикации, техническая документация и открытые проекты в области 3D-печати.

Ожидаемые научные результаты включают создание работоспособной системы управления, обеспечивающей:

* + - Чтение и обработку G-кода с SD-карты.
    - Синхронизацию четырех асинхронных двигателей по оси Z с использованием датчиков Холла.
    - Управление шаговыми двигателями по осям X и Y.
    - Надёжную подачу бетона и обработку аварийных ситуаций.

Практическая значимость работы заключается в разработке доступной и масштабируемой системы управления, пригодной для печати малогабаритных бетонных изделий. Использование недорогого микроконтроллера Arduino Uno и стандартных компонентов делает систему привлекательной для малых предприятий и образовательных учреждений. Научная значимость состоит в оптимизации алгоритмов синхронизации двигателей и обработки G-кода, что способствует развитию технологий строительной 3D-печати.

Положения, выносимые на защиту:

* Разработанная система управления на базе Arduino Uno обеспечивает точное позиционирование печатающей головки и синхронизацию асинхронных двигателей.
* Алгоритмы обработки G-кода и управления подачей бетона позволяют эффективно выполнять задачи печати малогабаритных изделий.
* Механизм обработки аварийных ситуаций повышает надёжность и безопасность работы принтера.

Апробация результатов проведена путем тестирования прототипа 3D-принтера, включающего выполнение тестовых заданий печати, обработку аварийных ситуаций. Результаты подтвердили работоспособность системы в лабораторных условиях.

# ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ АВТОМАТИЗИРОВАННОГО УПРАВЛЕНИЯ СТРОИТЕЛЬНЫМ 3D-ПРИНТЕРОМ

## Виды строительных 3D-принтеров

Любая 3D-печать основана на послойном нанесении какого-либо материала через печатающую головку, которая перемещается по заданным координатам, поэтому требуют определённой точности позиционирования и надежного управления.

Строительные 3D-принтеры различаются по конструкции, принципу перемещения печатающей головки и области применения. Основные виды реализации включают платформенные, крановые и роботизированные системы, каждая из которых имеет свои преимущества и ограничения [2].

**Крановые принтеры** используют подвижную стрелу или кран, на конце которого закреплена печатающая головка. Материал подается через шланг из внешнего хранилища или с помощью экструдера. Такие системы применяются для строительства крупных объектов, например, жилых домов или мостов, как в проектах компании Apis Cor [3]. Преимущества крановых принтеров — большой рабочий объем и мобильность, позволяющая работать на строительных площадках. Недостатки включают сложность управления, меньшую точность и высокую стоимость. На рисунке 1 показан внешний вид такого принтера.

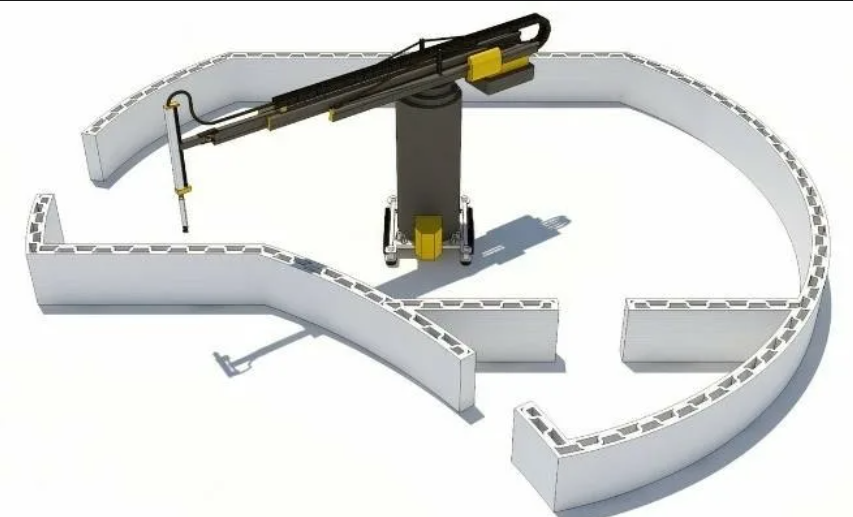


Рисунок 1. Пример кранового 3D-принтера

**Роботизированные принтеры** основаны на использовании роботизированных манипуляторов, которые обеспечивают многокоординатное движение печатающей головки. Такие системы применяются в сложных проектах, требующих высокой гибкости, например, при печати архитектурных форм с нестандартной геометрией. Их преимущества — универсальность и возможность интеграции с автоматизированными системами. Однако они требуют сложного программирования и дорогостоящего оборудования [4]. На рисунке 2 можно видеть пример такого принтера в работе [5].

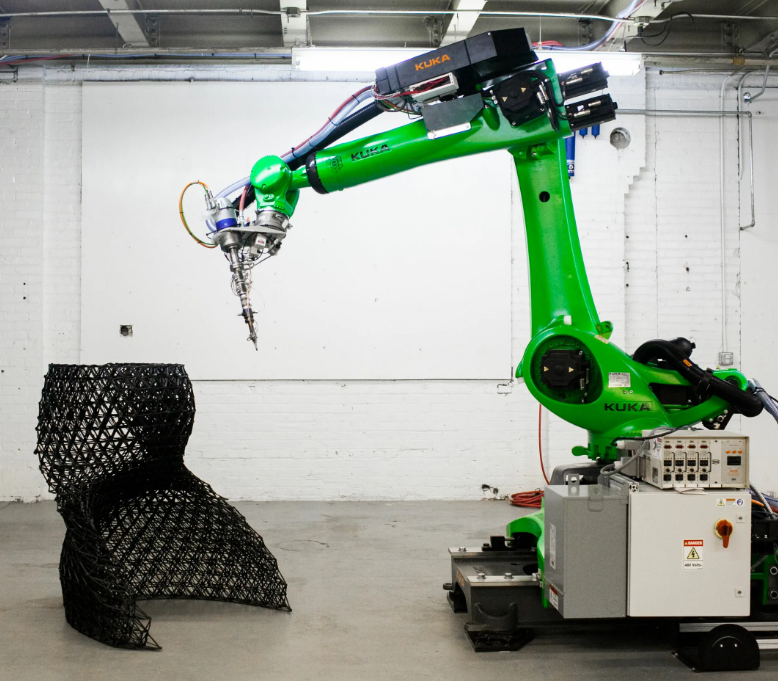


Рисунок 2. Пример роботизированного 3D-принтера

Платформенные принтеры используют стационарную раму или платформу, на которой размещены направляющие для перемещения печатающей головки по осям X, Y, Z. Платформа, как правило, поднимается или опускается с помощью двигателей, обеспечивая послойное нанесение материала. Такие системы, подобные разрабатываемой в данном проекте, подходят для печати как небольших изделий, так и многоэтажного строительства: всё определяется только длинной и платформы и высотой её подъёма. Их преимущества — простота конструкции, высокая точность позиционирования и экономическая доступность. На сегодняшний день являются наиболее перспективными для развития [6]. На рисунке 3 показан пример такого принтера [7].

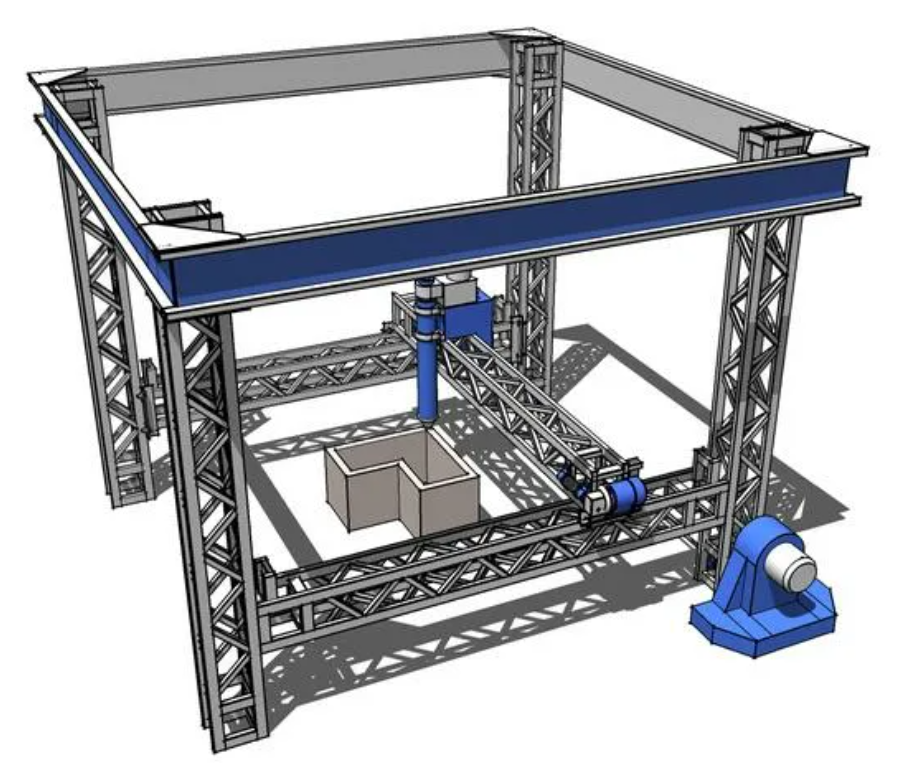


Рисунок 3. Пример платформенного 3D-принтера

## 1.2 Компоненты платформенных строительных 3D-принтеров

3D-принтер, используемый в данном проекте, является платформенным. Основные компоненты таких систем часто включают:

* Печатающую головку с соплом для подачи бетона.
* Платформу и двигатели на ней для перемещения печатающей головки по осям X, Y
* Двигатели для перемещения платформы по оси Z.
* Микроконтроллеры для обработки сигналов и управления движением.
* Датчики контроля положения и движения объектов системы.

В печатающую головку строительный материал (например, бетон) может подаваться как с внешнего хранилища через шланг, так и через экструдер (часть 3D-принтера, подающая материал для печати).

Для оси Z, отвечающей за подъем платформы, применяются асинхронные, шаговые, серво- и синхронные двигатели. Асинхронные двигатели, как в данном проекте, выбраны за мощность и низкую стоимость, с обратной связью для точности. Шаговые двигатели обеспечивают высокую точность, но не подходят для тяжелых нагрузок. Серво- и синхронные двигатели точны, но дороги и сложны в управлении, что избыточно для компактных принтеров. Сравнение представлено в таблице 1.

Таблица 1. Сравнение двигателей для оси Z

| Тип двигателя | Плюсы | Минусы | Применение в проекте |
| --- | --- | --- | --- |
| Асинхронный | Мощность, низкая стоимость, надежность | Низкая точность без обратной связи | Используется (ЭТФ-500 с датчиками Холла) |
| Шаговый | Высокая точность, простота управления | Ограниченная мощность, пропуск шагов | Не используется (применен для X, Y) |
| Серводвигатель | Высокая точность, мощность | Высокая стоимость, сложность | Не используется |
| Синхронный | Стабильность скорости, эффективность | Дороговизна, сложное управление | Не используется |

## 1.3 G-код в 3D-печати.

G-код (от английского “Geometric Code”) — это стандартный язык программирования, используемый для управления числовым программным управлением (ЧПУ) станками, включая 3D-принтеры. Он представляет собой набор инструкций, которые задают траекторию движения инструмента, параметры обработки и другие функции станка. В контексте 3D-печати G-код определяет последовательность действий экструдера, перемещения печатающей головки, температуру нагрева и другие параметры процесса печати.

G-код был разработан в 1950-х годах в Массачусетском технологическом институте (MIT) для управления первыми ЧПУ-станками. Изначально он использовался для фрезерных и токарных станков, где требовалось точно задавать перемещения инструмента в двух или трех измерениях. Стандарт был формализован в 1980-х годах как EIA RS-274-D [10], что позволило унифицировать команды для различных типов оборудования. С развитием аддитивных технологий G-код адаптировали для 3D-принтеров, где он стал основным языком управления, благодаря своей простоте и универсальности.

В 3D-печати G-код генерируется слайсерами - программами, которые преобразуют 3D-модель в набор инструкций для принтера. Примеры таких программ включают Cura, PrusaSlicer и Simplify3D. Слайсер анализирует геометрию модели, задает параметры печати (толщину слоя, заполнение, скорость) и формирует G-код, который затем передается на принтер.

G-код состоит из строк, каждая из которых — отдельная команда. Команды начинаются с буквы, обозначающей тип инструкции, за которой следуют числовые параметры. Основные категории команд:

* G-команды — задают геометрические действия (перемещение, дуговое движение, пауза).
* M-команды — управляют вспомогательными функциями (включение/выключение экструдера, вентилятора, нагрева).
* T-команды — выбор инструмента (в 3D-печати редко используются).
* Параметры — числовые значения, уточняющие действие (координаты X, Y, Z, скорость).

Пример строки G-кода для 3D-принтера: G1 X100 Y100 Z0.2 F3600 E0.5

Эта команда задает линейное перемещение (G1) печатающей головки в точку с координатами X=100 мм, Y=100 мм, Z=0.2 мм со скоростью 3600 мм/мин (F3600) и экструзией 0.5 мм материала (E0.5).

G-код для 3D-печати включает начальные команды (инициализация принтера, нагрев сопла и стола), основную часть (инструкции для печати модели) и завершающие команды (охлаждение, выключение).

В 3D-печати используется ограниченный набор G- и M-команд, адаптированных для аддитивных процессов. Основные команды приведены в таблице 2.

Таблица 2. Основные G- и M-команды, используемые в 3D-печати

|  |  |
| --- | --- |
| Команда | Описание |
| G0 | Быстрое перемещение без экструзии (позиционирование). |
| G1 | Линейное перемещение с экструзией (печать). |
| G28 | Возврат в начальную точку по одной или всем осям. |
| G90 | Установка абсолютного режима координат. |
| G91 | Установка относительного режима координат. |
| M03 | Начало вращения шпинделя по часовой стрелке |
| M05 | Конец вращения шпинделя |
| M140 | Установка температуры стола. |
| M190 | Установка температуры стола с ожиданием нагрева. |
| M106 | Включение вентилятора (с заданием скорости). |
| M107 | Выключение вентилятора. |

В текущем проекте строительной 3D-печати будут использоваться команды G1, G28, M3, M5.

# ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ

2.1 Принципы синхронизации однофазных асинхронных двигателей

Синхронизация четырёх однофазных асинхронных двигателей, располагающихся на «ногах» 3D-принтера, необходима для согласованного подъема платформы по оси Z. Синхронизация заключается в контроле числа оборотов каждого двигателя с помощью обратной связи. В данном проекте применяются датчики Холла, фиксирующие обороты вентилятора охлаждения (через прикреплённый на нём магнит), связанного с валом через редуктор с передаточным числом 50 к 1 (50 оборотов вентилятора равно одному обороту вала. Около 46 оборотов вентилятор совершает за одну секунду). Данные с датчиков о количестве оборотов вентилятора передаются на локальные микроконтроллеры Arduino, где программный счётчик считает количество срабатываний датчика, и отдаёт эти данные центральному Arduino при каждом его запросе.

Для передачи данных с локальных Arduino на центральный, находящуюся в электрическом щитке на расстоянии 1-10 м от «ног» принтера, необходима помехоустойчивая передача сигнала, для чего выбрана общая шина RS485, проходящая от плат Arduino через конверторы TTL-RS485, позволяющая передавать данные на расстоянии до 1200 метров [8], и протокол передачи данных Modbus RTU, обеспечивающий опрос ведущим устройством (центральный Arduino) ведомых (Arduino на «ногах») на общей шине с высокой скоростью без коллизий.

На рисунке 4 показана блок-схема обратной связи.

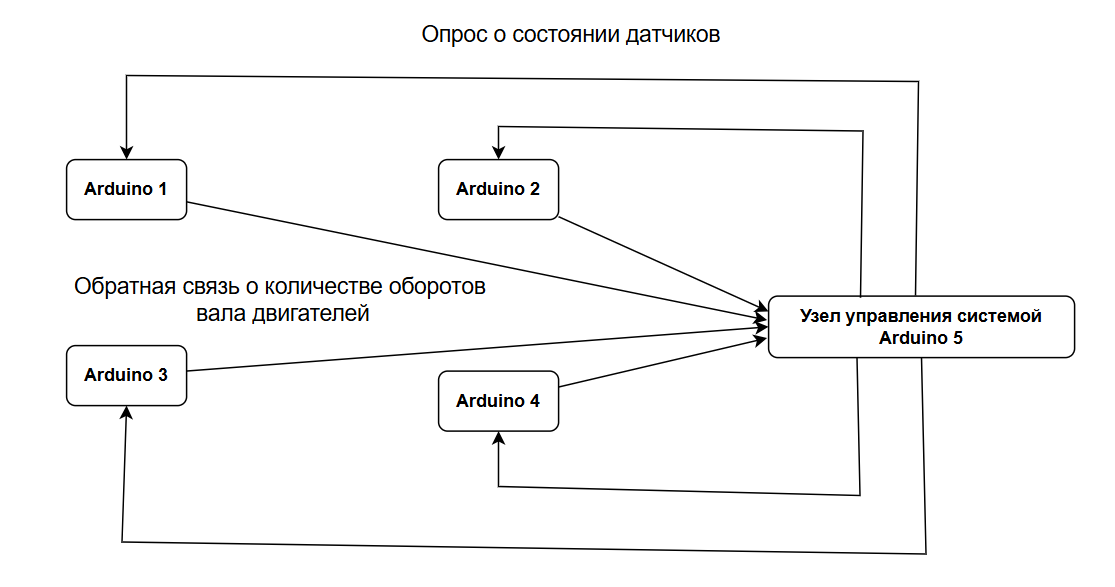


Рисунок 4. Блок-схема синхронизации двигателей

С локальных Arduino на «ногах» принтера данные обратной связи идут к центральному микроконтроллеру, который сравнивает обороты и отключает двигатель при достижении заданного значения.

## 2.2 Передача данных по протоколу Modbus RTU на общей шине RS485.

Modbus RTU (Remote Terminal Unit) — это широко используемый протокол передачи данных в системах промышленной автоматизации. Он работает по принципу ведущий-ведомый, где одно устройство (ведущее) инициирует обмен данными, а остальные (ведомые) отвечают на запросы, что исключает коллизии в сети. Протокол использует компактный бинарный формат, включающий адрес устройства, код функции, данные и контрольную сумму CRC, обеспечивая высокую помехоустойчивость и точность передачи. На физическом уровне чаще всего реализуется через шину RS485, которая позволяет подключать до 32 устройств на одну линию и, как уже говорилось пункте 1.3, поддерживает передачу данных на расстояния до 1200 метров.

Скорость передачи данных по протоколу Modbus RTU выбрана 19200 бод/c, так как она обеспечивает баланс между приемлемой для проекта частотой общения ведущего и ведомых устройств на шине и надежной, помехоустойчивой передачей данных на расстоянии до 10 метров.

Чтобы рассчитать наибольшую частоту опроса ведущим устройством ведомых при скорости передачи данных 19200 бод/с, рассмотрим структуру пакета данных, формат запроса ведущим устройством и формат ответа ведомым.

В Modbus RTU пакет состоит из:

* **Адрес** (1 байт).
* **Код функции** (1 байт).
* **Данные** (переменной длины).
* **CRC** (2 байта)

Для передачи с локального Arduino используем три регистра для надёжного хранения данных: два – на датчик Холла, один – на концевой датчик.

Формат запроса от ведущего устройства:

* Адрес: 1 байт (например, 0x01 для вашей ноги).
* Код функции: 1 байт (0x03 для чтения регистров).
* Начальный адрес регистра: 2 байта (например, 0x0000 для чтения с первого регистра).
* Количество регистров: 2 байта (0x0004 для чтения 3 регистров).
* CRC: 2 байта.

Итого: 1 + 1 + 2 + 2 + 2 = 8 байт

Формат ответа от ведомого устройства:

* Адрес: 1 байт (например, 0x01).
* Код функции: 1 байт (0x03).
* Количество байт данных: 1 байт (указывает, сколько байт данных следует дальше).
* Данные: N⋅2 байт, где N — количество регистров. Для 3 регистров: 3⋅2=6 байт.
* CRC: 2 байта.

Итого: 1 + 1 + 1 + 6 + 2 = 11 байт.

Время передачи одного пакета Modbus RTU (8 и 11 байт – но с запасом рассмотрим 20 байт) в одну сторону рассчитывается по формуле:

Tпакета = ≈  0,01 (с), где 10 бит - байт данных и стартовый, стоповый биты.

Для опроса 4 ведомых устройств (ног) время полного цикла:

Tцикл ≈ 4 ⋅ 0,01 = 0,04 (с), что позволяет опрашивать все 4 «ноги» принтера с частотой до 1 / 0,04 = 25 Гц.

На выбор конечной частоты опроса ведущим устройством ведомых влияет:

* допустимый перекос платформы оси Z, определяемый сохранением качества печати с этим перекосом.
* качество передачи данных по шине
* возможности центрального микроконтроллера, так как избыточная высокая частота опроса будет создавать лишнюю дополнительную нагрузку.

Сам небольшой допустимый перекос не окажет влияния на 3D-печать, так как каждый подъём и спуск платформы будет синхронизирован, то есть обороты вала вентилятора, «набежавшие» из-за задержек передачи и погрешности опроса, будут выравниваться по причине того, что центральный Arduino при каждом новом опросе получает актуальные данные о всех совершённых оборотах с локального Arduino на «ноге» принтера с самого начала печати и выравнивает дальнейшее движение платформы.

Шина RS485 при передаче данных по протоколу Modbus RTU, как было описано выше, поддерживает опрос четырёх «ног» до 25 Гц. Выбор платы центрального микроконтроллера будет описан в главе 2.2, где будут учтены требования к частоте опроса и в целом нагрузка на микроконтроллер. Сейчас для выбора частоты опроса остаётся определить только допустимый перекос платформы.

Время одного подъёма-спуска платформы во время печати составляет примерно 2 секунды, за которые вал двигателя совершит 1,84 оборота (46 ⋅ 2 = 92 оборота вала вентилятора при передаточном числе редуктора 1 к 50 это 0,92 ⋅ 2 = 1,84 оборота вала двигателя). Слой печати – около 30 миллиметров.

В традиционном строительстве допуски на отклонения защитного слоя бетона составляют 3–8 мм для слоев 10–50 мм [9]

Пусть для текущего проекта это будет 5 процентов отклонения платформы на 30 миллиметров слоя.

Рассчитаем лишние обороты вала вентилятора для 5 процентов отклонения платформы. Далее определим при какой частоте опроса ведущим устройством возможна такая погрешность в оборотах и выберем это частоту для шины:

Шаг подъёма двигателя составляет ≈  16,304 мм/оборот.

1 оборот вентилятора = 1 / 50 = 0,02 (оборота вала).

Подъём платформы при одном обороте вала вентилятора составляет 0,02 ⋅ 16,304 ≈ 0,32608 (мм).

Таким образом, для перекоса платформы 5 процентов (1,5 миллиметра печатного слоя из 30) допустимые обороты вала вентилятора составляют . Столько оборотов вентилятора пройдёт за 100 миллисекунд, так как за 1 секунду проходит 46 оборотов. Цикл в 100 миллисекунд – это частота 10 Герц.

Таким образом, частота опроса ведущим устройством ведомых будет выбрана 10 Герц и настроена шине.

## 2.3 Проектирование системы

Управление всеми элементами 3D-принтера осуществляется с центрального микроконтроллера Arduino Mega, находящегося в электрическом щитке вместе с другими электротехническими элементами: драйверами шаговых двигателей блоком питания, конденсаторами и другого (подробнее в главе 2.2).

Плата центрального микроконтроллера выбрана Arduino Mega, так как имелась в наличие, однако для задач проекта могла бы подойти, так же как для плат Modbus-ведомых устройств, более дешёвая Arduino Uno со специальной платой расширения портов.

Arduino Mega в щитке подключена к конвертору MAX485, преобразующего сигнал TTL от Arduino в RS485 для помехоустойчивой передачи, от которого идут провода 1-7 метров к опорам принтера в такие же конверторы и от них в Arduino Uno.



Рисунок 5. Общий вид 3D-принтера

Движение сопла в осях X и Y происходит за счёт установленных на принтере шаговых двигателей, получающих сигналы управления от драйверов из щитка.

На каждой опоре, или «ноге», принтера установлены асинхронные однофазные двигатели с тросом и крюком, закреплённым за верх опоры, для движения платформы по оси Z. На каждом двигателе имеется вентилятор охлаждения, соединённый с валом двигателя через редуктор с передаточным числом 1 к 50, куда прикреплён круглый неодимовый магнит. На сам же двигатель – датчик Холла, фиксирующий прохождение магнита (то есть полный оборот вентилятора), и подающего сигнал на Arduino Uno на опоре.

На каждой опоре рядом с двигателем примонтирована распаячная коробка с микроконтроллером Arduino Uno внутри, который фиксирует срабатывания датчика Холла, программно ведёт счётчик этих срабатываний и отправляет эти данные через конвертор MAX485 на центральный Arduino Mega. На рисунке 6 показана опора принтера. На ней закреплён асинхронный двигатель, для подъёма платформы, снизу – шаговый двигатель для движения по оси Y, рядом – распаячная коробка с платой Arduino Uno.



Рисунок 6. Опора 3D-принтера

Управление движением каждым из четырёх асинхронных двигателей ведётся с центрального Arduino Mega через твёрдотелые реле, два на каждый двигатель (движение вверх, движение вниз). Когда с Arduino Uno на опоре приходят данные о том, что двигатель уже совершил необходимое количество оборотов, работа этого двигателя прекращается. Так достигается синхронизация, и платформа всегда оказывается в ровном положении с заранее вычисленным допустимым для печати перекосом (подробнее описано в главе 1.4).

Два из четырёх локальных Arduino Uno, помимо количества срабатываний с датчика Холла, принимают также данные о срабатывании концевых датчиков, располагающихся на каждой опоре принтера в их нижнем положении для определения начала координат по оси Z, а для осей X и Y ещё два концевых датчика на первой и второй «ноге» соответственно, при получении кода G28 («поиск нуля»).

В электрический щиток питание поступает с трёхфазной сети через трёхфазный автоматический выключатель (для обеспечения необходимого тока). Далее, одна фаза поступает на первые два асинхронных двигателя, вторая – на два других, а третья – на блок питания (так же в щитке) 60 Вольт для питания драйверов шаговых двигателей, питание Arduino и двигатель в сопле для подачи бетона. Каждая линия фазы сперва проходит через автоматические выключатели соответствующего номинала.

# ГЛАВА 3. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ

## 3.1 Аппаратная реализация

В текущем проекте системы управления строительным 3D-принтером есть 2 аппаратных блока: центральный (электрический ящик с главным микроконтроллером и другим оборудованием) и локальный (монтажные коробки с Arduino Uno, закреплённые на четырёх опорах принтера, и подключённые к ним датчики).

В приложении А перечислены все элементы управления 3D-принтером и их описание. **Итого**: 16 компонентов, 53 единиц элементов управления. Исходя из приложения А потребляемая мощность принтера составляет около 6.43 кВт.

На рисунке 7 показана принципиальная схема узла управления, а в таблице 3 обозначения к схеме.

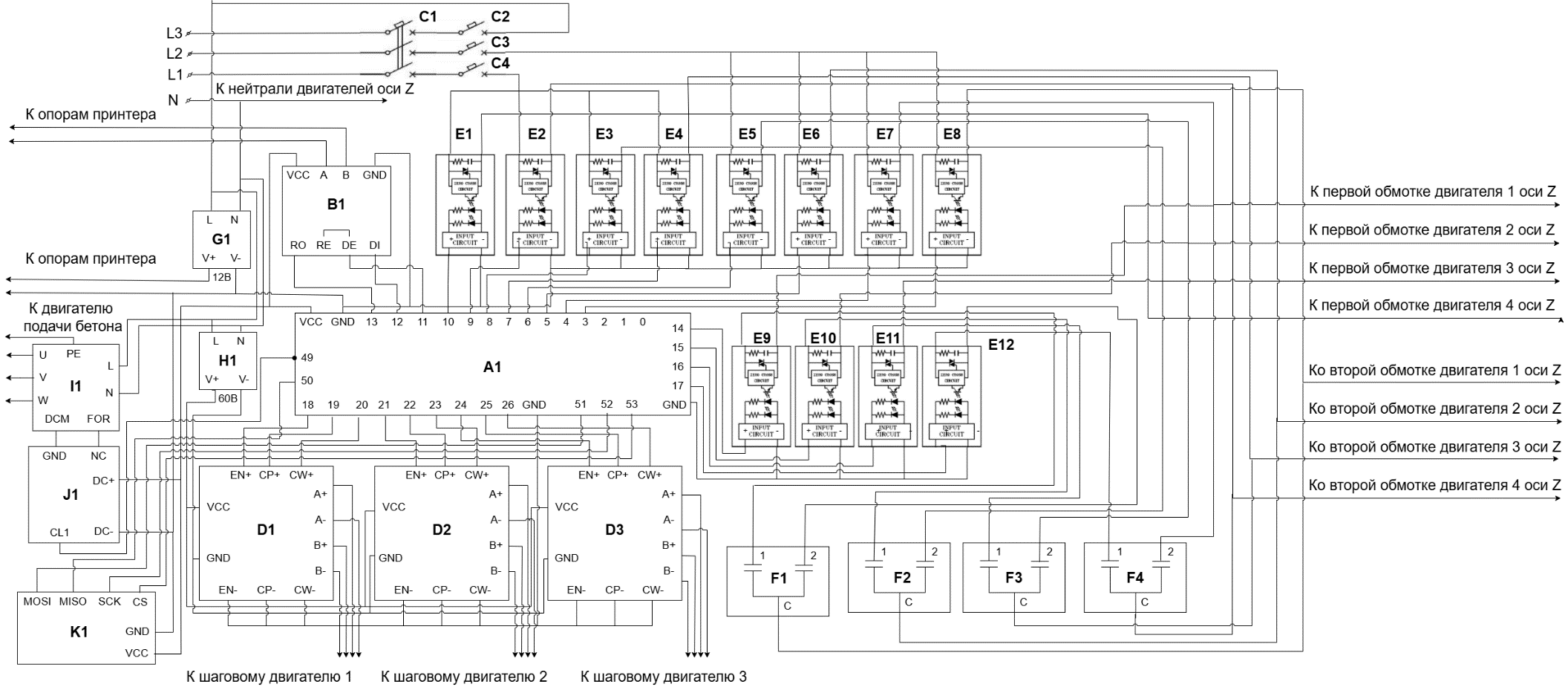


Рисунок 7. Принципиальная схема узла управления

Таблица 3. обозначений к принципиальной схеме электрического щитка

| Обозначение | Количество, шт. | Наименование | Примечание |
| --- | --- | --- | --- |
| A1 | 1 | Микроконтроллер Arduino Mega 2560 | Центральное управление, опрос по Modbus RTU, управление шаговыми двигателями |
| B1 | 1 | Конвертор TTL-RS485 MAX485 | Передача данных по шине RS485, ручное управление направлением передачи |
| С1 | 1 | Автоматический выключатель трёхфазный DEKraft C25 | Защита цепи, 230/400В |
| С2 | 3 | Автоматический выключатель однофазный DEKraft C25 | Защита цепи, 230В |
| D1–D3 | 3 | Драйвер шагового двигателя CW8060 | Управление шаговыми двигателями HB86HS114-4600А, ток до 6 А |
| E1–E12 | 12 | Твердотельное реле SSR-40DA | E1–E8: управление двигателями ЭТФ-500; E9–E12: разрыв пускового конденсатора |
| F1–F4 | 1 | Конденсатор LMG CBB60 | Рабочие и пусковой конденсаторы в одном корпусе, 10 и 35 мкФ, для ЭТФ-500 |
| G1 | 1 | Блок питания YG-24W | Питание Arduino Uno и датчиков на опорах, 12В, 2А |
| H1 | 1 | Блок питания S-400-60 | Питание шаговых двигателей HB86HS114-4600А, 60 В, 6.7 А |
| I1 | 1 | Частотный преобразователь HY01D523B | Управление двигателем подачи бетона |
| J1 | 1 | Реле-модуль RM1LS | Замыкание контактов частотного преобразователя через Arduino |
| K1 | 1 | Модуль SD-карты HW-203 | Подключение SD-карты к Arduino (SPI) |

На рисунке 8 представлен внешний вид электрического шкафа.



Рисунок 8. Вид внутри электрического щитка

Для шины RS485 выбран кабель CAT 5e, обеспечивающий помехоустойчивую передачу за счёт витой пары, то есть дифференциальной передачи сигнала, что является стандартом для шины. Для согласования нулевых потенциалов на шине порт GND Arduino Mega 2560 соединён с минусом блока питания YG-24W, питающем Arduino Uno и конверторы MAX485 на опорах принтера.

Пусковые конденсаторы, согласно документации к асинхронным двигателям ЭТФ-500, должны отключаются после входа в рабочий режим двигателя. Выбрано время 500 миллисекунд после каждого запуска (через реле E9-E12).

На рисунке 10 показана принципиальная схема для опоры принтера, включающая датчик Холла, концевой датчик и конвертор для шины RS485.

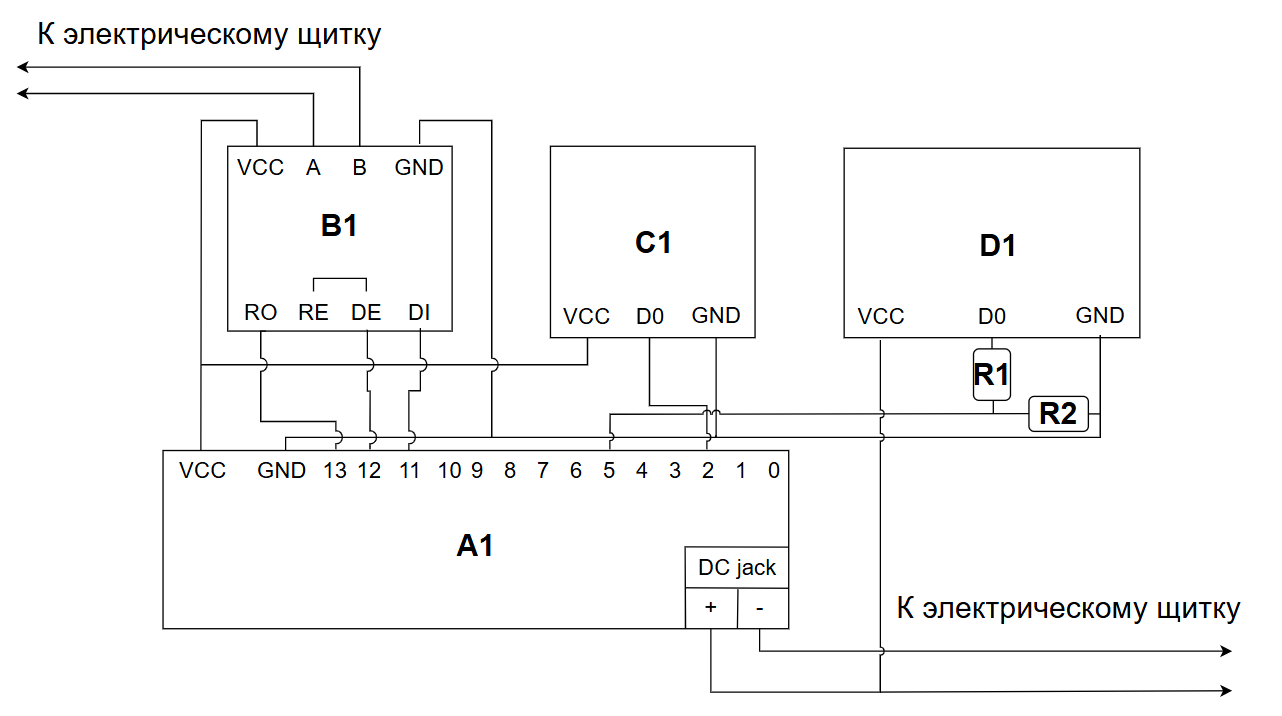


Рисунок 9. Принципиальная схема для опоры принтера

В таблице 4 представлены обозначения для принципиальной схемы опоры принтера:

Таблица 4. Элементы принципиальной схемы щитка

| Обозначение | Количество, шт. | Наименование | Примечание |
| --- | --- | --- | --- |
| A1 | 1 | Микроконтроллер Arduino Uno | Фиксируем срабатывания датчиков, по Modbus RTU отправляет на центральный микроконтроллер |
| B1 | 1 | Конвертор TTL-RS485 MAX485 | Передача данных по шине RS485, ручное управление направлением передачи |
| С1 | 1 | Датчик Холла MH-Sensor Series | Подсчёт количества оборотов вала вентилятор охлаждения асинхронного двигателя |
| D1 | 1 | Концевой датчик LJ12A3-4-Z/BY | Инициализация начала координат в осях X, Y, Z |
| R1-R2 | 2 | Резисторы 20 (R1) и 10 (R2) кОм | Делитель напряжения для цифрового выхода концевого датчика |

Датчик Холла подсоединяется к порту 2 Arduino Uno для инициализации аппаратных прерываний при получении сигнала для более эффективного приёма данных с датчика.

Концевой датчик получает питание 12В от блока питания YG-24W из щитка, поэтому его цифровой выход так же имеет напряжение 12В, неприемлемое для цифровых пинов Arduino. Для приведения напряжения до уровня 4В используются делитель напряжения из двух резисторов 10 и 20 кОм, как показано на рисунке 10.

Итоговое напряжение, приходящее на Arduino Uno, вычисляется по формуле [10]:

Uвых = = ,

где R1 = 20 кОм, R2 = 10 кОм.

## 3.2 Программная реализация

Программная часть системы управления строительным 3D-принтером разработана для обработки G-кода и координации движений печатающей головки в трехмерном пространстве. Она реализована в среде Arduino IDE и включает, как уже было описано в главе 2, центральную плату Arduino Mega, который управляет шаговыми двигателями по осям X, Y и печатающей головкой, а также четыре локальных Arduino Uno, отвечающих за синхронизацию асинхронных двигателей по оси Z.

G-код поступает с SD-карты и обрабатывается центральным Arduino Mega. Для обработки G-кодов для текущего проекта написана библиотека motion\_control.cpp. Она разбирает команды G-кода, например, G1 X100 Y50 Z10 F1000, и преобразует их либо в последовательность импульсов для драйверов шаговых двигателей, либо в сигналы на реле реверсирования асинхронными двигателями оси Z, либо в частотный преобразователь управления для управления двигателем подачи бетона. Имеется также обработка аварийных ситуаций таких, как потеря связи с ведомым.

Программа разделена на модули, каждый из которых отвечает за определённую функцию системы:

* sd\_card.cpp — инициализация SD-карты и чтение G-кода.
* gcode\_parser.cpp — разбор G-кода и формирование команд для выполнения.
* motion\_control.cpp — выполнение движений и управление подачей бетона.
* hall\_sensor.cpp — обработка сигналов датчиков Холла и концевых датчиков.
* motor\_control.cpp — управление асинхронными двигателями через твердотельные реле.
* printer\_state.cpp — управление состоянием принтера (IDLE, PRINTING, EMERGENCY).
* modbus\_config.cpp — настройка протокола Modbus RTU для связи между Arduino Mega и Arduino Uno.
* pins\_arduino.cpp — конфигурация пинов Arduino Mega.

Основной файл gCodeParser\_SD.ino объединяет все модули, реализуя главный цикл программы. Саму логику программы можно представить в виде блок-схемы, показанной а рисунке 10.

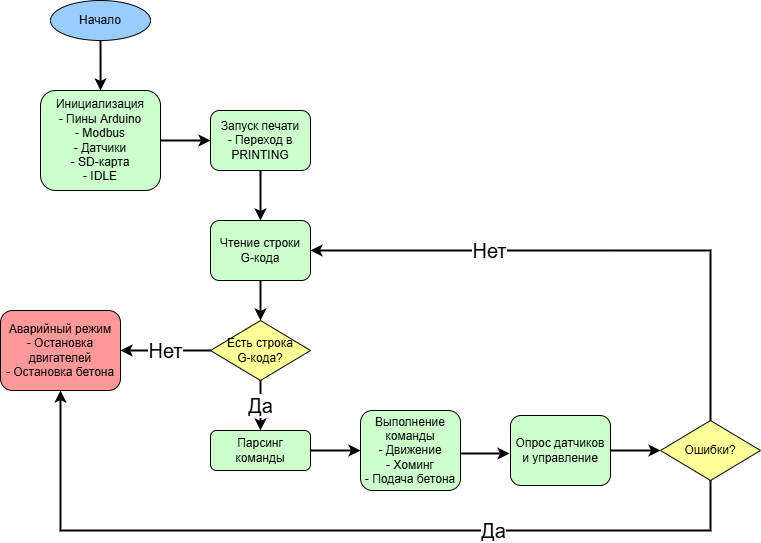


Рисунок 10. Блок-схема программы

Для печати принтер читает G-код из файла gcode.txt на SD-карте. Инициализация SD-карты выполняется в функции initSDCard() с использованием пина CS (53). Если карта или файл недоступны, программа выводит сообщение об ошибке и останавливается.

void initSDCard() {

Serial.print("Инициализация SD-карты... Пин CS=");

Serial.println(SD\_CS);

if (!SD.begin(SD\_CS)) {

Serial.println("Ошибка инициализации SD-карты!");

while (1);

}

gcodeFile = SD.open("gcode.txt");

if (!gcodeFile) {

Serial.println("Ошибка открытия gcode.txt!");

while (1);

}

}

G-код построчно считывается функцией getNextGCodeLine() и передаётся в parseGCodeLine() для разбора. Парсер извлекает тип команды (G или M), номер команды и параметры (X, Y, Z, F). Например, команда G1 X100 Y50 Z10 F100 интерпретируется как линейное движение к точке (100, 50, 10) со скоростью 100 мм/мин. Поддерживаются команды:

* G0/G1 — линейное движение.
* G28 — хоминг (возврат в нулевую позицию).
* M3/M5 — включение/выключение подачи бетона.

После разбора команда передаётся в executeGCodeCommand() для выполнения.

Четыре асинхронных двигателя ЭТФ-500 на опорах принтера управляют движением по оси Z. Для синхронизации используются датчики Холла, которые фиксируют обороты вентилятора (50 срабатываний = 1 оборот вала). Локальные Arduino Uno обрабатывают сигналы датчиков через прерывания, что исключает пропуск импульсов:

void hollSensorISR() {

if (!digitalRead(HOLL\_SENSOR\_PIN)) {

hollSensorCount++;

}

}

attachInterrupt(digitalPinToInterrupt(HOLL\_SENSOR\_PIN), hollSensorISR, FALLING);

Данные о срабатываниях передаются по Modbus RTU (скорость 38400 бод, таймаут 50 мс) на Arduino Mega с частотой 10 Гц. Функция pollHallSensors() опрашивает все четыре опоры, обновляя счётчики hallCounts и позиции hallPosition. Коэффициент пересчёта срабатываний датчика Холла в миллиметры составляет 0.5 (1 мм = 0.5 срабатывания).

При движении по Z двигатели запускаются функциями startMotorsUp() или startMotorsDown(), а остановка происходит, когда текущая позиция (hallPosition) достигает целевой. Для предотвращения перегрузки пусковые конденсаторы отключаются через 500 мс после старта:

void checkCapacitorTimers() {

if (capacitorDisablePending && millis() - capacitorTimer >= STARTCAP\_OFF\_TIME) {

disableStartCapacitors();

Serial.println("Пусковые конденсаторы отключены");

}

}

Поиск нуля (команда G28) выполняется в функции homeAxes(). Процесс включает:

* Движение к X=0, Y=0.
* Проверку концевых датчиков. Если хотя бы один сработал, принтер поднимается на 70 мм для предотвращения столкновения.
* Движение вниз до срабатывания всех концевых датчиков.
* Сброс позиций (X=0, Y=0, Z=0).
* Таймаут хоминга составляет 30 секунд. Если он истекает, система переходит в аварийное состояние (EMERGENCY).
* Движение по осям X и Y

Шаговые двигатели HB86HS114-4600А управляют движением печатающей головки по осям X и Y через драйверы CW8060. Функция moveTo() вычисляет количество шагов (33 шага/мм) и генерирует импульсы на пины STEP\_X и STEP\_Y:

int32\_t stepsX = (x - currentX) \* STEPS\_PER\_MM;

digitalWrite(DIR\_X, stepsX >= 0 ? LOW : HIGH);

for (int32\_t i = 0; i < abs(stepsX); i++) {

digitalWrite(STEP\_X, HIGH);

delayMicroseconds(X\_DELAY);

digitalWrite(STEP\_X, LOW);

delayMicroseconds(X\_DELAY);

}

Задержка 250 мкс обеспечивает плавное движение. Движение по X и Y выполняется последовательно с движением по Z для упрощения синхронизации.

Подача бетона включается командой M3 и выключается командой M5. Сигнал подаётся на пин CONCRETE\_SUPPLY через реле. Для предотвращения преждевременной подачи между включением и началом печати выдерживается задержка 1000 миллисекунд.

void startConcreteSupply() {

digitalWrite(CONCRETE\_SUPPLY, LOW);

Serial.println("Подача бетона ВКЛ");

}

Система поддерживает состояние EMERGENCY, которое активируется при:

* Ошибке связи по Modbus (2 последовательных сбоя).
* Истечении таймаута хоминга или движения.
* Ошибке инициализации SD-карты.

В аварийном состоянии все двигатели останавливаются, а подача бетона отключается. Функция enterEmergencyState() логирует причину сбоя.

Для коммуникации по протоколу Modbus RTU используется библиотека ModbusMaster для Arduino Mega и SimpleModbusSlaveSoftwareSerial для Arduino Uno. Задержка между опросами (10 мс) оптимизирована для скорости 38400 бод.

Все ключевые действия (движение, хоминг, ошибки) логируются в Serial для отладки.

Программная реализация обеспечивает полный цикл 3D-печати: от чтения G-кода до синхронизированного управления двигателями и подачей бетона. Код оптимизирован для надёжности и диагностики, а модульная структура позволяет легко расширять функциональность, например, для добавления поддержки новых команд G-кода.

## 3.3 Тестирование системы

Система успешно выполняет печать по G-коду, синхронизируя движение платформы по оси Z и печатающей головки по осям X и Y. Реализована надёжная обработка аварийных ситуаций, таких как сбои связи по Modbus или таймауты «хоминга».

Проведённое тестирование подтвердило стабильность работы при выполнении тестовых G-кодов для печати простых геометрических форм. На рисунке 11 показан результат тестовой печати квадрата длиной стороны 250 миллиметров и 4 слоями.

**

Рисунок 11. Тестовая печать

На рисунке 12 представлен один из асинхронных двигателей оси Z с датчиком Холла и магнитом, прикреплённым к вентилятору охлаждения.

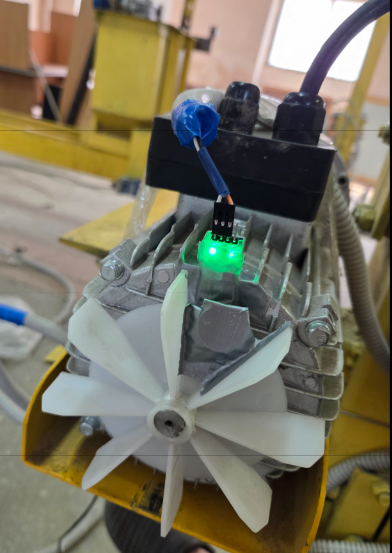


Рисунок 12. Срабатывание датчика Холла на двигателе

На рисунке 13 показано срабатывание концевого датчика, сигнализирующего о том, что по его опоре платформа пришла в начало координат.

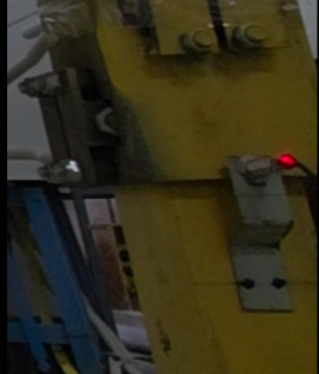


Рисунок 13. Срабатывание концевого датчика

# ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы разработана полноценная автоматизированная система управления строительным 3D-принтером, способная выполнять печать бетонных изделий, таких как элементы мебели, декора и малые архитектурные формы, на основе G-кода.

Выполненные задачи:

* Архитектура системы: Спроектирована и реализована система на базе центрального микроконтроллера Arduino Mega 2560, управляющего четырьмя локальными Arduino Uno, установленными на опорах принтера. Связь между контроллерами осуществляется по протоколу Modbus RTU через шину RS485.
* Аппаратная часть: Настроена и протестирована электроника, включая твердотельные реле SSR-40DA, пусковые конденсаторы LPMG CBB60, датчики Холла, концевые датчики и драйверы шаговых двигателей CW8060. Полный перечень компонентов приведён в приложении А.
* Программное обеспечение: разработано в среде Arduino IDE и включает:
* Чтение и обработку G-кода с SD-карты для управления движением и подачей бетона.
* Синхронизацию четырёх асинхронных двигателей ЭТФ-500 по оси Z с использованием датчиков Холла, обеспечивающих точность позиционирования.
* Управление шаговыми двигателями HB86HS114-4600А по осям X и Y через драйверы CW8060.
* Реализацию хоминга (G28) с использованием концевых датчиков для предотвращения столкновений.
* Управление подачей бетона через частотный преобразователь.
* Обработку аварийных ситуаций с переходом в состояние EMERGENCY при сбоях связи, таймаутах или ошибках SD-карты.
* Модульность кода: Программа разделена на модули (sd\_card.cpp, gcode\_parser.cpp, motion\_control.cpp и другие), что упрощает отладку и масштабирование.

Для дальнейшего совершенствования системы предполагаются следующие доработки:

* Безопасность: установить дополнительные концевые датчики на осях X и Y для предотвращения столкновений с краями платформы.
* Надёжность: добавить плату (например, Arduino Nano) в качестве супервизора для мониторинга состояния Arduino Mega и перезапуска системы при сбоях.
* Сохранение состояния: интегрировать микросхему EEPROM для сохранения текущей позиции и параметров печати при сбоях питания, что позволит возобновлять печать.
* Энергозащита: включить в схему электрического щитка источник бесперебойного питания (ИБП) для обеспечения времени на запись текущей позиции в EEPROM в случае отключений электроэнергии.
* Расширенный функционал: добавить поддержку дополнительных G-команд (например, G2/G3 для дугового движения) и возможность управления скоростью подачи бетона через ШИМ-сигнал.
* Интерфейс: реализовать пользовательский интерфейс (например, на базе LCD-экрана или веб-приложения) для мониторинга процесса печати и ввода параметров.

Разработанный 3D-принтер может использоваться для производства бетонных изделий в строительной отрасли, включая элементы мебели, декора и малые архитектурные формы. Система также подходит для образовательных целей как пример ЧПУ-станка, демонстрирующего принципы автоматизации и управления. В перспективе возможно масштабирование системы для печати более крупных конструкций при доработке механической части и увеличении мощности двигателей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Habr. Строительная 3D-печать в ожидании прорыва // Хабр. – URL: https://habr.com/ru/articles/673542/ (дата обращения: 01.05.2025).
2. Gectaro. 3D-печать в строительстве: технология применения и как это работает // Gectaro. – URL: https://gectaro.com/blog/tpost/zv8tsz1j61-3d-pechat-v-sfere-stroitelstva-korotko-o (дата обращения: 02.05.2025).
3. Additiv Tech. Из первых уст: рассказ инженера Apis Cor о строительстве рекордного 3D-печатного здания в Дубае // Additiv Tech. – URL: https://additiv-tech.ru/publications/iz-pervyh-ust-rasskaz-inzhenera-apis-cor-o-stroitelstve-rekordnogo-3d-pechatnogo (дата обращения: 03.05.2025).
4. Дзен. 3D-печать роботизированной рукой – полное руководство // Дзен. – URL: https://dzen.ru/a/ZIeI-qLmTij6ypTg (дата обращения: 05.05.2025).
5. Pinterest. Пример роботизированного 3D-принтера // Pinterest. – URL: https://ru.pinterest.com/pin/how-3d-printers-will-lower-prices-make-fantasies-real-and-transform-the-housing-market--92605336067672391/ (дата обращения: 07.05.2025).
6. Easy3DPrint. 3D-принтеры для строительства: особенности и преимущества // Easy3DPrint. – URL: https://easy3dprint.com.ua/ru/3d-printery-dlya-stroitelstva/ (дата обращения: 10.05.2025).
7. 3Dtoday. Встреча с разработчиками 3D-принтера для печати бетоном в Томске // 3Dtoday. – URL: https://3dtoday.ru/blogs/mikhashev/meeting-with-the-developers-of-3d-printer-to-print-concrete-in-tomsk (дата обращения: 13.05.2025).
8. RobotChip. Обзор конвертера TTL-RS485 // RobotChip. – URL: https://robotchip.ru/obzor-konvertera-ttl-rs485/ (дата обращения: 13.05.2025).
9. Tekla. Определения толщины защитного слоя армирования // Tekla Structures. – URL: https://support.tekla.com/ru/doc/tekla-structures/2020/rei\_cover\_thickness (дата обращения: 13.05.2025).
10. Стандарт EIA RS-274-D. Формат данных переменного блока для позиционирования, контурной обработки и числового управления станками / Electronic Industries Association. — Вашингтон, США: Electronic Industries Association, 1979. — 200 с.
11. Алейник А.С., Востриков Е.В., Волковский С.А., Дейнека И.Г., Стригалев В.Е., Мешковский И.К. Основы схемотехники приемопередающих электронных устройств: учебно-методическое пособие. – СПб.: Университет ИТМО, 2021. – 149 с.

# ПРИЛОЖЕНИЕ А

**Таблица А.1. Компоненты системы управления 3D-принтером.**

| 1 | Компонент | Количество, шт. | Модель | Назначение | Характеристики |
| --- | --- | --- | --- | --- | --- |
| 1 | Плата | 5 | Arduino Uno (4), Arduino Mega 2560 (1) | Общение по Modbus RTU, счётчик датчиков, Центральное управление | ATmega2560, 16 МГц, 54 цифровых пина. ATmega328P, 8-бит, 16 МГц, 14 цифровых пина |
| 2 | Конвертор TTL-RS485 | 5 | MAX485 | Передача данных по шине RS485 | Скорость до 2,5 Мбит/с, расстояние до 1200 м |
| 3 | Однофазный двигатель | 4 | ЭТФ-500 | Подъем платформы по оси Z через тельфер | Мощность 1 кВт, 220 В, 2800 об/мин |
| 4 | Трёхфазный двигатель | 1 | АИР63В2 | Подача бетона | Мощность 1 кВт, 3000 об/мин |
| 5 | Частотный преобразователь | 1 | HY01D523B | Управление двигателем подачи бетона | Мощность 1.5 кВт, 220В, выходная частота 400 Гц |
| 6 | Модуль SD-карты | 1 | HW-203 | Подключение SD-карты к Arduino | Интерфейс SPI |
| 7 | Реле-Модуль | 1 | RM1LS | Замыкание контактов частотного преобразователя через Arduino Mega | Вход 5VDC, Выход 30 VDC |
| 8 | Шаговый двигатель | 3 | HB86HS114-4600А | Позиционирование головки по осям X и Y | 6 А, 1,8°/шаг, момент 8,5 Н·м |
| 9 | Датчик Холла | 4 | MH-sensor series | Контроль оборотов вентилятора | Цифровой выход, 5 VDC |
| 10 | Концевой датчик | 6 | LJ12A3-4-Z/BY | Инициализация начала координат в осях X, Y, Z | Напряжение: DC 6–36 VAC, выходной ток: 300 мА, частота отклика: 0,5 кГц |
| 11 | Твердотельное реле | 12 | SSR-40DA | Управление двигателями (8 шт.) и разрыв пускового конденсатора (4 шт.) | 24–380 VAC, ток до 40 А |
| 12 | Конденсатор | 4 | LMG CBB60 | Рабочие и пусковой конденсаторы для двигателей ЭТФ-500 | 35 и 10 мкФ, 450 VAC |
| 13 | Автоматический выключатель | 1 | DEKraft C25 | Защита цепи от перегрузки | Номинальный ток 25 А, 230/400 VAC |
| 14 | Драйвер шагового двигателя | 3 | CW8060 | Управление шаговыми двигателями | Вход: 24-80 VDC |
| 15 | Блок питания (драйвера) | 1 | WAW S-400-60 | Питание драйверов шаговых двигателей | Вход: АС 220 VAC, выход: 60 VDC (до 6.7 А) |
| 16 | Блок питания (Arduino) | 1 | YG-24W | Питание плат Arduino | Вход: АС 220 VAC, выход: 12 VDC (до 2 А) |

# ПРИЛОЖЕНИЕ Б

Листинги программ.

Листинг Б.1. Код программы Modbus-ведомого устройства для Arduino Uno.

#include <SimpleModbusSlaveSoftwareSerial.h>

#include <SoftwareSerial.h>

// Настройка программного UART для MAX485

SoftwareSerial rs485(9, 12);

// Пины датчиков и DE/RE

#define END\_SENSOR\_PIN 10

#define HOLL\_SENSOR\_PIN 2

#define DE\_RE\_PIN 7 // Пин для DE/RE

constexpr uint8\_t slaveAddress { 3 }; // Адрес слейва (1, 2, 3 или 4)

// Регистры для хранения данных датчиков

enum {

  END\_SENSOR\_COUNT\_HIGH = 0,

  END\_SENSOR\_COUNT\_LOW = 1,

  HOLL\_SENSOR\_COUNT\_HIGH = 2,

  HOLL\_SENSOR\_COUNT\_LOW = 3,

  TOTAL\_REGS\_SIZE

};

uint16\_t holdingRegs[TOTAL\_REGS\_SIZE];

volatile uint32\_t endSensorCount = 0;

volatile uint32\_t hollSensorCount = 0;

bool lastEndSensorState = HIGH;

void hollSensorISR() {

  if (!digitalRead(HOLL\_SENSOR\_PIN)) {

    hollSensorCount++;

  }

}

void setup() {

  // Настройка пина DE/RE

  pinMode(DE\_RE\_PIN, OUTPUT);

  digitalWrite(DE\_RE\_PIN, LOW); // По умолчанию прием

  // Настройка RS485 через SoftwareSerial

  rs485.begin(19200);

  // Инициализация Modbus Slave

  modbus\_configure(&rs485, 19200, slaveAddress, DE\_RE\_PIN, TOTAL\_REGS\_SIZE);

Продолжение листинга Б.1

  // Настройка пинов датчиков

  pinMode(END\_SENSOR\_PIN, INPUT\_PULLUP);

  pinMode(HOLL\_SENSOR\_PIN, INPUT);

  // Инициализация регистров

  for (int i = 0; i < TOTAL\_REGS\_SIZE; i++) {

    holdingRegs[i] = 0;

  }

// Настройка прерывания для датчика Холла

  attachInterrupt(digitalPinToInterrupt(HOLL\_SENSOR\_PIN), hollSensorISR, FALLING);

}

void loop() {

  // Программный опрос концевого датчика

  bool currentEndSensorState = digitalRead(END\_SENSOR\_PIN);

  if (currentEndSensorState != lastEndSensorState && currentEndSensorState == LOW) {

    endSensorCount++;

  }

  lastEndSensorState = currentEndSensorState;

// Обновление Modbus-регистров

  holdingRegs[END\_SENSOR\_COUNT\_HIGH] = (endSensorCount >> 16) & 0xFFFF;

  holdingRegs[END\_SENSOR\_COUNT\_LOW] = endSensorCount & 0xFFFF;

  holdingRegs[HOLL\_SENSOR\_COUNT\_HIGH] = (hollSensorCount >> 16) & 0xFFFF;

  holdingRegs[HOLL\_SENSOR\_COUNT\_LOW] = hollSensorCount & 0xFFFF;

  // Обработка запросов Modbus

  modbus\_update(holdingRegs);

}

Листинг Б.2. Код программы Modbus-мастера для Arduino Mega (gCodeParser\_SD.ino)

#include "gcode\_parser.h"

#include "motion\_control.h"

#include "printer\_state.h"

#include "motor\_control.h"

#include "hall\_sensor.h"

#include "modbus\_config.h"

#include "pins\_arduino.h"

#include "sd\_card.h"

void setup() {

Serial.begin(115200);

while (!Serial);

setupArduinoPins();

setupModbus();

resetHallCounts(0);

initGCodeParser();

setPrinterState(PrinterState::IDLE);

Serial.println("Принтер инициализирован");

}

void loop() {

if (getPrinterState() != PrinterState::EMERGENCY) {

pollHallSensors(getPrinterState() == PrinterState::PRINTING);

}

checkCapacitorTimers();

switch (getPrinterState()) {

case PrinterState::IDLE:

setPrinterState(PrinterState::PRINTING);

Serial.println("Начало печати");

break;

case PrinterState::PRINTING:

GCodeCommand cmd;

if (getNextGCodeCommand(&cmd)) {

executeGCodeCommand(cmd);

Serial.print("Выполняется: ");

Serial.print(cmd.type);

Serial.print(cmd.number);

Serial.print(" X=");

Serial.print(cmd.x);

Serial.print(" Y=");

Serial.print(cmd.y);

Serial.print(" Z=");

Продолжение листинга Б.2

Serial.print(cmd.z);

Serial.print(" F=");

Serial.print(cmd.f);

Serial.print(" Concrete=");

Serial.println(cmd.concrete);

} else {

setPrinterState(PrinterState::EMERGENCY);

stopConcreteSupply();

Serial.println("Печать завершена, переход в EMERGENCY");

}

break;

case PrinterState::PAUSED:

break;

case PrinterState::EMERGENCY:

break;

}

}

Б.3. Код модуля обработки SD-карты (sd\_card.cpp)

#include "sd\_card.h"

#include "pins\_arduino.h"

#include <SPI.h>

File gcodeFile;

void initSDCard() {

Serial.print("Инициализация SD-карты... Пин CS=");

Serial.println(SD\_CS);

if (!SD.begin(SD\_CS)) {

Serial.println("Ошибка инициализации SD-карты!");

while (1);

}

Serial.println("SD-карта инициализирована.");

gcodeFile = SD.open("gcode.txt");

if (!gcodeFile) {

Serial.println("Ошибка открытия gcode.txt!");

while (1);

}

Serial.println("Файл gcode.txt открыт");

}

String getNextGCodeLine() {

String line = "";

if (gcodeFile.available()) {

line = gcodeFile.readStringUntil('\n');

line.trim();

} else {

gcodeFile.close();

}

return line;

}

#include "gcode\_parser.h"

#include "sd\_card.h"

static bool parserInitialized = false;

void initGCodeParser() {

if (!parserInitialized) {

initSDCard();

parserInitialized = true;

}

}

bool getNextGCodeCommand(GCodeCommand\* cmd) {

String line = getNextGCodeLine();

if (line == "") {

Продолжение листинга Б.3

if (gcodeFile) {

gcodeFile.seek(0);

}

return false;

}

return parseGCodeLine(line, cmd);

}

bool parseGCodeLine(String line, GCodeCommand\* cmd) {

cmd->type = 0;

cmd->number = -1;

cmd->x = NAN;

cmd->y = NAN;

cmd->z = NAN;

cmd->f = NAN;

cmd->concrete = false;

cmd->homing = false;

line.trim();

if (line.length() == 0 || line.startsWith(";")) {

return false;

}

if (line[0] == 'G' || line[0] == 'M') {

cmd->type = line[0];

int spaceIdx = line.indexOf(' ');

String numStr = (spaceIdx == -1) ? line.substring(1) : line.substring(1, spaceIdx);

cmd->number = numStr.toInt();

} else {

return false;

}

if (cmd->type == 'G' && cmd->number == 28) {

cmd->homing = true;

cmd->x = 0.0;

cmd->y = 0.0;

cmd->z = NAN;

}

int pos = line.indexOf(' ') + 1;

while (pos > 0 && pos < line.length()) {

char param = line[pos];

String valueStr = "";

pos++;

while (pos < line.length() && line[pos] != ' ') {

valueStr += line[pos];

pos++;

}

Продолжение листинга Б.3

float value = valueStr.toFloat();

switch (param) {

case 'X': cmd->x = value; break;

case 'Y': cmd->y = value; break;

case 'Z': cmd->z = value; break;

case 'F': cmd->f = value; break;

}

pos = line.indexOf(' ', pos) + 1;

}

if (cmd->type == 'M') {

if (cmd->number == 3) {

cmd->concrete = true;

} else if (cmd->number == 5) {

cmd->concrete = false;

}

}

return true;}

Б.5. Код модуля конфигурации пинов Arduino (pins\_arduino.cpp)

#include "pins\_arduino.h"

#include "modbus\_config.h"

void setupArduinoPins() {

pinMode(SSR11, OUTPUT);

digitalWrite(SSR11, LOW);

pinMode(SSR12, OUTPUT);

digitalWrite(SSR12, LOW);

pinMode(SSR21, OUTPUT);

digitalWrite(SSR21, LOW);

pinMode(SSR22, OUTPUT);

digitalWrite(SSR22, LOW);

pinMode(SSR31, OUTPUT);

digitalWrite(SSR31, LOW);

pinMode(SSR32, OUTPUT);

digitalWrite(SSR32, LOW);

pinMode(SSR41, OUTPUT);

digitalWrite(SSR41, LOW);

pinMode(SSR42, OUTPUT);

digitalWrite(SSR42, LOW);

pinMode(SSR8\_1, OUTPUT);

digitalWrite(SSR8\_1, HIGH);

pinMode(SSR7\_2, OUTPUT);

digitalWrite(SSR7\_2, HIGH);

pinMode(SSR5\_3, OUTPUT);

digitalWrite(SSR5\_3, HIGH);

pinMode(SSR6\_4, OUTPUT);

digitalWrite(SSR6\_4, HIGH);

pinMode(DIR\_Y, OUTPUT);

pinMode(STEP\_Y, OUTPUT);

pinMode(DIR\_X, OUTPUT);

pinMode(STEP\_X, OUTPUT);

pinMode(CONCRETE\_SUPPLY, OUTPUT);

digitalWrite(CONCRETE\_SUPPLY, HIGH);

pinMode(DE\_RE\_PIN, OUTPUT);

pinMode(SD\_CS, OUTPUT);

digitalWrite(SD\_CS, HIGH);

}

Б.6. Код модуля управления движением (motion\_control.cpp)

#include "motion\_control.h"

#include "printer\_state.h"

#include "hall\_sensor.h"

void executeGCodeCommand(GCodeCommand cmd) {

Serial.print("Received command: ");

Serial.print(cmd.type);

Serial.print(cmd.number);

Serial.print(" X=");

Serial.print(cmd.x);

Serial.print(" Y=");

Serial.print(cmd.y);

Serial.print(" Z=");

Serial.print(cmd.z);

Serial.print(" F=");

Serial.println(cmd.f);

if (cmd.type == 'G') {

if (cmd.number == 0 || cmd.number == 1) {

float targetX = isnan(cmd.x) ? currentX : cmd.x;

float targetY = isnan(cmd.y) ? currentY : cmd.y;

float targetZ = isnan(cmd.z) ? currentZ : cmd.z;

float feedrate = isnan(cmd.f) ? 100.0 : cmd.f;

moveTo(targetX, targetY, targetZ, feedrate);

} else if (cmd.number == 28) {

Serial.println("Calling homeAxes");

homeAxes();

Serial.println("homeAxes completed");

}

} else if (cmd.type == 'M') {

if (cmd.number == 3) {

startConcreteSupply();

delay(DELAY\_BEFORE\_CONCRETE\_SUPPLY);

Serial.println("Delay after M3 completed");

} else if (cmd.number == 5) {

stopConcreteSupply();

}

}

}

void homeAxes() {

Serial.println("Хоминг ...");

Serial.println("Движение к X=0, Y=0");

moveTo(0.0, 0.0, currentZ, 100.0);

Serial.println("Подготовка хоминга по Z...");

unsigned long homingStart = millis();

pollHallSensors(false);

Продолжение листинга Б.6

bool allEndstopsZero = true;

uint32\_t totalHallCounts = 0;

for (uint8\_t i = 0; i < 4; i++) {

totalHallCounts += getHallCount(i);

if (getEndSensorCount(i) >= 1) {

allEndstopsZero = false;

Serial.print("Слейв ");

Serial.print(i + 1);

Serial.println(" концевик сработал (EndS>=1)");

}

}

if (!allEndstopsZero && totalHallCounts == 0) {

Serial.println("Подъем Z на 80 мм");

moveTo(currentX, currentY, currentZ + LIFTING\_Z\_BEFORE\_HOMING\_INCASE\_ENDS1, 100.0);

Serial.println("Ожидание 2 секунд после подъема");

delay(2000);

}

Serial.println("Хоминг по Z...");

startMotorsDown();

delay(500);

uint32\_t initialEndCounts[4];

pollHallSensors(false);

for (uint8\_t i = 0; i < 4; i++) {

initialEndCounts[i] = getEndSensorCount(i);

Serial.print("Слейв ");

Serial.print(i + 1);

Serial.print(" начальный EndS=");

Serial.println(initialEndCounts[i]);

}

bool homed[4] = {false, false, false, false};

while (!(homed[0] && homed[1] && homed[2] && homed[3])) {

pollHallSensors(false);

checkCapacitorTimers();

for (uint8\_t i = 0; i < 4; i++) {

if (!homed[i] && getEndSensorCount(i) >= initialEndCounts[i] + 1) {

stopMotor(i + 1);

motorStopped[i] = true;

homed[i] = true;

Serial.print("Слейв ");

Serial.print(i + 1);

Serial.println(" хоминг по Z завершен (концевик сработал)");

}

Продолжение листинга Б.6

}

if (millis() - homingStart > 30000) {

Serial.println("Таймаут хоминга");

for (uint8\_t i = 0; i < 4; i++) {

if (!homed[i]) {

stopMotor(i + 1);

motorStopped[i] = true;

}

}

enterEmergencyState(0, "Таймаут хоминга по Z");

return;

}

}

Serial.println("Ожидание 2 секунд для стабилизации");

delay(2000);

resetHallCounts(0);

currentX = 0.0;

currentY = 0.0;

currentZ = 0.0;

Serial.println("Хоминг завершен: X=0, Y=0, Z=0");

}

void moveTo(float x, float y, float z, float feedrate) {

if (z != currentZ) {

bool allMotorsStopped = false;

bool motorStoppedLocal[4] = {false, false, false, false};

unsigned long moveStart = millis();

for (uint8\_t i = 0; i < 4; i++) {

motorStopped[i] = false;

}

if (z > currentZ) {

startMotorsUp();

delay(500);

while (!allMotorsStopped && (millis() - moveStart < 30000)) {

pollHallSensors(true);

checkCapacitorTimers();

allMotorsStopped = true;

for (uint8\_t i = 0; i < 4; i++) {

if (!motorStoppedLocal[i]) {

float currentPos = getHallPosition(i);

if (currentPos >= z) {

stopMotor(i + 1);

Продолжение листинга Б.6

motorStoppedLocal[i] = true;

motorStopped[i] = true;

Serial.print("Мотор ");

Serial.print(i + 1);

Serial.println(" остановлен (вверх)");

} else {

allMotorsStopped = false;

}

}

}

}

} else {

startMotorsDown();

delay(500);

while (!allMotorsStopped && (millis() - moveStart < 30000)) {

pollHallSensors(false);

checkCapacitorTimers();

allMotorsStopped = true;

for (uint8\_t i = 0; i < 4; i++) {

if (!motorStoppedLocal[i]) {

float currentPos = getHallPosition(i);

if (currentPos <= z) {

stopMotor(i + 1);

motorStoppedLocal[i] = true;

motorStopped[i] = true;

Serial.print("Мотор ");

Serial.print(i + 1);

Serial.println(" остановлен (вниз)");

} else {

allMotorsStopped = false;

}

}

}

}

}

if (!allMotorsStopped) {

for (uint8\_t i = 0; i < 4; i++) {

if (!motorStoppedLocal[i]) {

stopMotor(i + 1);

motorStopped[i] = true;

}

}

enterEmergencyState(0, "Таймаут движения по Z");

return;

}

Продолжение листинга Б.6

for (uint8\_t i = 0; i < 4; i++) {

hallPosition[i] = z \* HALL\_COUNTS\_PER\_MM;

lastHallCounts[i] = hallCounts[i];

Serial.print("Слейв ");

Serial.print(i + 1);

Serial.print(" позиция установлена на Z=");

Serial.println(z);

}

currentZ = z;

}

int32\_t stepsX = (x - currentX) \* STEPS\_PER\_MM;

int32\_t stepsY = (y - currentY) \* STEPS\_PER\_MM;

int32\_t maxSteps = max(abs(stepsX), abs(stepsY));

digitalWrite(DIR\_X, stepsX >= 0 ? LOW : HIGH);

digitalWrite(DIR\_Y, stepsY >= 0 ? LOW : HIGH);

for (int32\_t i = 0; i < maxSteps; i++) {

if (i < abs(stepsX)) {

digitalWrite(STEP\_X, HIGH);

delayMicroseconds(X\_DELAY);

digitalWrite(STEP\_X, LOW);

delayMicroseconds(X\_DELAY);

}

if (i < abs(stepsY)) {

digitalWrite(STEP\_Y, HIGH);

delayMicroseconds(Y\_DELAY);

digitalWrite(STEP\_Y, LOW);

delayMicroseconds(Y\_DELAY);

}

}

currentX = x;

currentY = y;

}

void startConcreteSupply() {

digitalWrite(CONCRETE\_SUPPLY, LOW);

Serial.println("Подача бетона ВКЛ");

}

void stopConcreteSupply() {

digitalWrite(CONCRETE\_SUPPLY, HIGH);

Serial.println("Подача бетона ВЫКЛ");

}

Листинг Б.7. Код модуля обработки датчиков Холла (hall\_sensor.cpp)

#include "hall\_sensor.h"

#include "printer\_state.h"

uint32\_t hallCounts[4] = {0, 0, 0, 0};

static uint32\_t lastHallCounts[4] = {0, 0, 0, 0};

static uint32\_t hallPosition[4] = {0, 0, 0, 0};

static uint32\_t endSensorCounts[4] = {0, 0, 0, 0};

static uint8\_t slaveFailCount[4] = {0, 0, 0, 0};

bool motorStopped[4] = {false, false, false, false};

void pollHallSensors(bool isMovingUp) {

unsigned long startCycle = millis();

for (uint8\_t slaveID = 1; slaveID <= 4; slaveID++) {

Serial.print("Slave ");

Serial.print(slaveID);

Serial.print(": ");

node.begin(slaveID, rs485);

uint8\_t result = node.readHoldingRegisters(0x00, 4);

uint8\_t motorIndex = slaveID - 1;

if (result == node.ku8MBSuccess) {

slaveFailCount[motorIndex] = 0;

endSensorCounts[motorIndex] = ((uint32\_t)node.getResponseBuffer(0) << 16) | node.getResponseBuffer(1);

uint32\_t count = ((uint32\_t)node.getResponseBuffer(2) << 16) | node.getResponseBuffer(3);

hallCounts[motorIndex] = count;

if (!motorStopped[motorIndex]) {

int32\_t delta = count - lastHallCounts[motorIndex];

if (delta != 0) {

if (isMovingUp) {

if (hallPosition[motorIndex] <= UPPER\_Z\_LIMIT - delta) {

hallPosition[motorIndex] += delta;

} else {

hallPosition[motorIndex] = UPPER\_Z\_LIMIT;

}

} else {

hallPosition[motorIndex] = (hallPosition[motorIndex] > delta) ? hallPosition[motorIndex] - delta : 0;

}

}

lastHallCounts[motorIndex] = count;

}

Serial.print("EndS=");

Продолжение листинга Б.7

Serial.print(endSensorCounts[motorIndex]);

Serial.print(", Count=");

Serial.print(hallCounts[motorIndex]);

Serial.print(", Pos=");

Serial.println(getHallPosition(motorIndex));

} else {

slaveFailCount[motorIndex]++;

Serial.println("N/A");

if (slaveFailCount[motorIndex] >= MODBUS\_MAX\_FAILS) {

enterEmergencyState(slaveID, "не ответил 2 раза");

return;

}

}

delay(MODBUS\_POLL\_DELAY);

}

unsigned long cycleTime = millis() - startCycle;

Serial.print("Cycle time: ");

Serial.print(cycleTime);

Serial.println(" ms");

}

uint32\_t getHallCount(uint8\_t motorID) {

return hallCounts[motorID];

}

float getHallPosition(uint8\_t motorID) {

return (float)hallPosition[motorID] / HALL\_COUNTS\_PER\_MM;

}

uint32\_t getEndSensorCount(uint8\_t motorID) {

return endSensorCounts[motorID];

}

void resetHallCounts(uint32\_t initialPosition) {

for (uint8\_t i = 0; i < 4; i++) {

hallCounts[i] = initialPosition \* HALL\_COUNTS\_PER\_MM;

lastHallCounts[i] = hallCounts[i];

hallPosition[i] = initialPosition \* HALL\_COUNTS\_PER\_MM;

endSensorCounts[i] = 0;

slaveFailCount[i] = 0;

motorStopped[i] = false;

}

}

Продолжение листинга Б.7

void enterEmergencyState(uint8\_t slaveID, const char\* reason) {

setPrinterState(PrinterState::EMERGENCY);

Serial.print("ЧС: слейв ");

Serial.print(slaveID);

Serial.print(": ");

Serial.println(reason);

}

Листинг Б.8. Код модуля управления состоянием принтера (printer\_state.cpp)

#include "printer\_state.h"

#include "motor\_control.h"

static PrinterState currentState = PrinterState::IDLE;

void setPrinterState(PrinterState state) {

if (state == PrinterState::EMERGENCY) {

for (uint8\_t i = 1; i <= 4; i++) {

stopMotor(i);

}

Serial.println("Emergency state activated");

}

currentState = state;

}

PrinterState getPrinterState() {

return currentState;

}

Листинг Б.9. Код модуля конфигурации Modbus (modbus\_config.cpp)

#include "modbus\_config.h"

#include "pins\_arduino.h"

SoftwareSerial rs485(RS485\_RX, RS485\_TX);

ModbusMaster node;

void preTransmission() {

digitalWrite(DE\_RE\_PIN, HIGH);

}

void postTransmission() {

digitalWrite(DE\_RE\_PIN, LOW);

}

void setupModbus() {

Serial.begin(SOFTWARESERIAL\_SPEED);

while (!Serial);

digitalWrite(DE\_RE\_PIN, LOW);

rs485.begin(MODBUS\_SPEED);

node.begin(1, rs485);

node.setTimeout(MODBUS\_TIMEOUT);

node.preTransmission(preTransmission);

node.postTransmission(postTransmission);

}

Листинг Б.10. Код модуля управления двигателями (motor\_control.cpp)

#include "motor\_control.h"

#include "pins\_arduino.h"

static unsigned long capacitorTimer = 0;

static bool capacitorDisablePending = false;

void startMotorsUp() {

digitalWrite(SSR12, LOW);

digitalWrite(SSR22, LOW);

digitalWrite(SSR32, LOW);

digitalWrite(SSR42, LOW);

enableStartCapacitors();

digitalWrite(SSR11, HIGH);

digitalWrite(SSR21, HIGH);

digitalWrite(SSR31, HIGH);

digitalWrite(SSR41, HIGH);

capacitorTimer = millis();

capacitorDisablePending = true;

Serial.println("startMotorsUp called, capacitor timer started");

}

void startMotorsDown() {

digitalWrite(SSR11, LOW);

digitalWrite(SSR21, LOW);

digitalWrite(SSR31, LOW);

digitalWrite(SSR41, LOW);

enableStartCapacitors();

digitalWrite(SSR12, HIGH);

digitalWrite(SSR22, HIGH);

digitalWrite(SSR32, HIGH);

digitalWrite(SSR42, HIGH);

capacitorTimer = millis();

capacitorDisablePending = true;

Serial.println("startMotorsDown called, capacitor timer started");

}

void stopMotor(uint8\_t motorID) {

switch (motorID) {

case 1:

digitalWrite(SSR11, LOW);

digitalWrite(SSR12, LOW);

break;

case 2:

digitalWrite(SSR21, LOW);

digitalWrite(SSR22, LOW);

break;

Продолжение листинга Б.10

case 3:

digitalWrite(SSR31, LOW);

digitalWrite(SSR32, LOW);

break;

case 4:

digitalWrite(SSR41, LOW);

digitalWrite(SSR42, LOW);

break;

}

}

void disableStartCapacitors() {

digitalWrite(SSR8\_1, LOW);

digitalWrite(SSR7\_2, LOW);

digitalWrite(SSR5\_3, LOW);

digitalWrite(SSR6\_4, LOW);

capacitorDisablePending = false;

Serial.println("disableStartCapacitors called");

}

void enableStartCapacitors() {

digitalWrite(SSR8\_1, HIGH);

digitalWrite(SSR7\_2, HIGH);

digitalWrite(SSR5\_3, HIGH);

digitalWrite(SSR6\_4, HIGH);

Serial.println("enableStartCapacitors called");

}

void checkCapacitorTimers() {

if (capacitorDisablePending && millis() - capacitorTimer >= STARTCAP\_OFF\_TIME) {

disableStartCapacitors();

Serial.println("Пусковые конденсаторы отключены");

}

}

Листинг Б.11. Код заголовочного файла конфигурации Modbus (modbus\_config.h)

#ifndef MODBUS\_CONFIG\_H

#define MODBUS\_CONFIG\_H

#include <ModbusMaster.h>

#include <SoftwareSerial.h>

#define SOFTWARESERIAL\_SPEED 115200

#define MODBUS\_SPEED 38400

#define MODBUS\_TIMEOUT 50

#define MODBUS\_POLL\_DELAY 10

#define MODBUS\_MAX\_FAILS 2

#define STARTCAP\_OFF\_TIME 500

#define UPPER\_Z\_LIMIT 1000000000

#define DELAY\_BEFORE\_CONCRETE\_SUPPLY 1000

#define HOMING\_TIMEOUT 100000000000000000

const uint32\_t LIFTING\_Z\_BEFORE\_HOMING\_INCASE\_ENDS1 = 70;

const uint32\_t STEPS\_PER\_POLL = 330;

const float HALL\_COUNTS\_PER\_MM = 0.5;

static uint32\_t initialPositionOffset = 0;

static float currentX = 0.0;

static float currentY = 0.0;

static float currentZ = 0.0;

const uint32\_t X\_DELAY = 250;

const uint32\_t Y\_DELAY = 250;

const uint32\_t STEPS\_PER\_MM = 33;

extern SoftwareSerial rs485;

extern ModbusMaster node;

extern bool posReset[4];

extern uint8\_t pollCount[4];

extern uint32\_t lastCount[4];

extern uint32\_t hallCounts[4];

extern bool isHoming;

void preTransmission();

void postTransmission();

void setupModbus();

#endif