МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт энергетики и электротехники
(наименование института полностью)

Кафедра « Промышленная электроника »
(наименование кафедры)
27.03.04 Управление в технических системах
(код и наименование направления подготовки, специальности)
Системы и технические средства автоматизации и управления
(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему Автоматизированный сверлильный станок для производства печатных плат

Студент	Н.А.Мальцев (И.О. Фамилия)	(личная подпись)
Руководитель	Е.С. Глибин (И.О. Фамилия)	(личная подпись)
Консультант	Н.В. Ященко (И.О. Фамилия)	(личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н., доцент А.А. Шевцов_____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____»_____20_____г.

Тольятти 2018

Аннотация

Объем 45 с., 33 рис., 1 табл., 20 источников.

Цель бакалаврской работы состоит в том, чтобы разработать программное обеспечение для контроллера, управляющему электрическими приводами сверлильного станка в ручном и автоматическом режимах.

Сам станок состоит из следующих частей: Arduino MEGA 2560, блока питания, четырёхстрочного дисплея с модулем I2C, драйвера CNC Shield V3.0, трёх трёхпозиционных переключателей, одного двухпозиционного переключателя, четырёх концевых переключателей и кнопки питания.

Сверлильный станок работает в ручном и автоматическом режимах. В ручном режиме он управляется оператором с помощью 3 переключателей и 1 кнопки. Первый переключатель “влево-вправо” для перемещения рабочего стола по оси X, второй – “вперёд-назад” для перемещения по оси Y, третий – “вверх-вниз” для перемещения шпинделя по оси Z и кнопка для включения/выключения двигателя шпинделя. В автоматическом режиме управление рабочим столом осуществляется с помощью G-кодов, передающихся через USB-интерфейс программой на ПК.

Также программа выводит на дисплей координаты нахождения шпинделя над рабочим столом.

Abstract

The topic of the given graduation work is Automatic Drilling Machine For the Production of Printed Circuit Boards.

The graduation work consist of an explanatory note on 45 pages, including 1 table, 33 figures, 5 foreign sources, 1 appendices and the graphic part on 6 A1 sheets.

The machine consists of the Arduino MEGA 2560, a power supply, a four-line display with an I2C module, driver CNC Shield V3.0, a set of 3-position switches, a two-position switch, a power button and 4 limit switches. Also in the work, we consider creation of manual and automatic control for a drilling machine.

In manual mode, control carried out with the help of the drill control switches “up-down” in the Z coordinate, “right-left” along the X coordinate and “forward-backward” along the Y coordinate. In automatic mode, we load G-code through the computer, which processed on the Arduino hardware platform and will give a signal for the operation of stepper motors and a display. Then the machine starts work.

Also the program displays the current direction of work, the coordinates of the drill over the desktop and the information that occurred on error.

In conclusion, we have created software for machine operation in manual and automatic modes.

Содержание

Введение 5

1. Обзор существующих решений 6

2. Структурная схема 9

3. Схема электрических соединений 11

4. Разработка алгоритма работы программы контроллера в ручном режиме 12

4.1 Блок-схема алгоритма работы программы контроллера в ручном режиме .. 12

4.2 Создание управляющей программы для ручного режима работы 14

5. G-код и автоматический режим 27

5.1 G-код 27

5.2 Блок-схема алгоритма работы программы контроллера в автоматическом режиме 28

5.3 Создание управляющей программы для автоматического режима работы . 30

6. Экспериментальное тестирование и отладка программы 34

7. Безопасность и экологичность проекта 41

8. Экономическая эффективность 42

Заключение 43

Список используемой литературы 44

Введение

В современном мире печатные платы – это физическая основа для электрического и механического соединений в электронной технике. Практически в любом предприятии, которое занимается производством электроники и электронных компонентов, требуется механическая обработка печатных плат для добавления переходных и монтажных отверстий. Из-за большого количества требуемых отверстий в печатных платах и высокой точности обработки появляется необходимость в автоматизированном сверлильном станке.

Помимо базовых функций любого сверлильного станка, таких как вращение и перемещение в вертикальном направлении режущего инструмента, в нем должна быть внедрена система автоматизации, так как это значительно повысит эффективность и точность обработки. Для автоматизации многие производители систем управления используют G-code, потому что он имеет жесткую структуру и используется на большинстве станках с ЧПУ.

1. Обзор существующих решений

В настоящее время существует довольно много автоматизированных сверлильных станков для производства печатных плат с разными режимами управления, составляющими компонентами, контроллерами и программным обеспечением к ним. Также существуют учебные стенды, которые служат для разработки и сбора различных видов манипуляторов для задач захвата и перемещения деталей. Они дают практические навыки разработки принципиальных электрических схем и программирования контроллеров.

Станция сверления CAMOZZI

Станция сверления входит в учебный стенд “Автоматизированная производственная ячейка DID-APL”. На рисунке 1.1 показана станция сверления отдельно от учебного стенда.

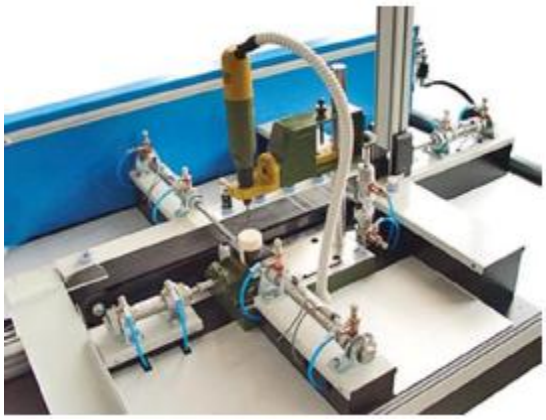

Рисунок 1.1 – Станция сверления Camozzi

Станция сверления состоит из следующих компонентов:

- Электрический шпиндель;
- Пневматические тиски;
- Четыре пневматических цилиндра;
- Два моностабильных распределителя клапанного типа;
- Бистабильный распределитель;

- Четыре магнитных датчика положений цилиндра;
- Реле давления.

Станок с ЧПУ Steepline

Станок с ЧПУ Steepline имеет рабочую поверхность 30х20 см, которая используется для закрепления печатных плат. Перемещение инструмента происходит вдоль цилиндрических направляющих. Точность позиционирования осуществляется шаровинтовыми парами. При этом точность положения сверла составляет 0,05 мм. Цилиндрические направляющие в станке используются для повышения надёжности и для быстрого перемещения сверла при серийном изготовлении плат. Также в станок встроена водяная система охлаждения, которая предотвращает перегревы и позволяет непрерывно производить печатные платы. Станок с ЧПУ Steepline изображен на рисунке 1.2.

Рисунок 1.2 – Сверлильный станок для производства печатных плат с ЧПУ Steepline

Перемещения шпинделя по оси Z у данного станка составляет 100 мм. Тип передачи по осям X, Y, Z осуществляется с помощью шаровинтовых пар, где используются 16мм винты с шагом 5 мм. Для перемещения используются шаговые двигатели 57HS. Также станок содержит в себе датчик длины инструмента и индуктивный датчик для определения начала нулевых точек станка. Мощность шпинделя составляет 0.8 кВт. При этом габаритный размер станка 620x500x560 мм и вес 58 кг. Средняя цена на данный станок 21.05.2018 составляет 88 тысяч рублей.

2. Структурная схема

На рисунке 2.1 представлена структурная схема автоматизированного сверлильного станка.

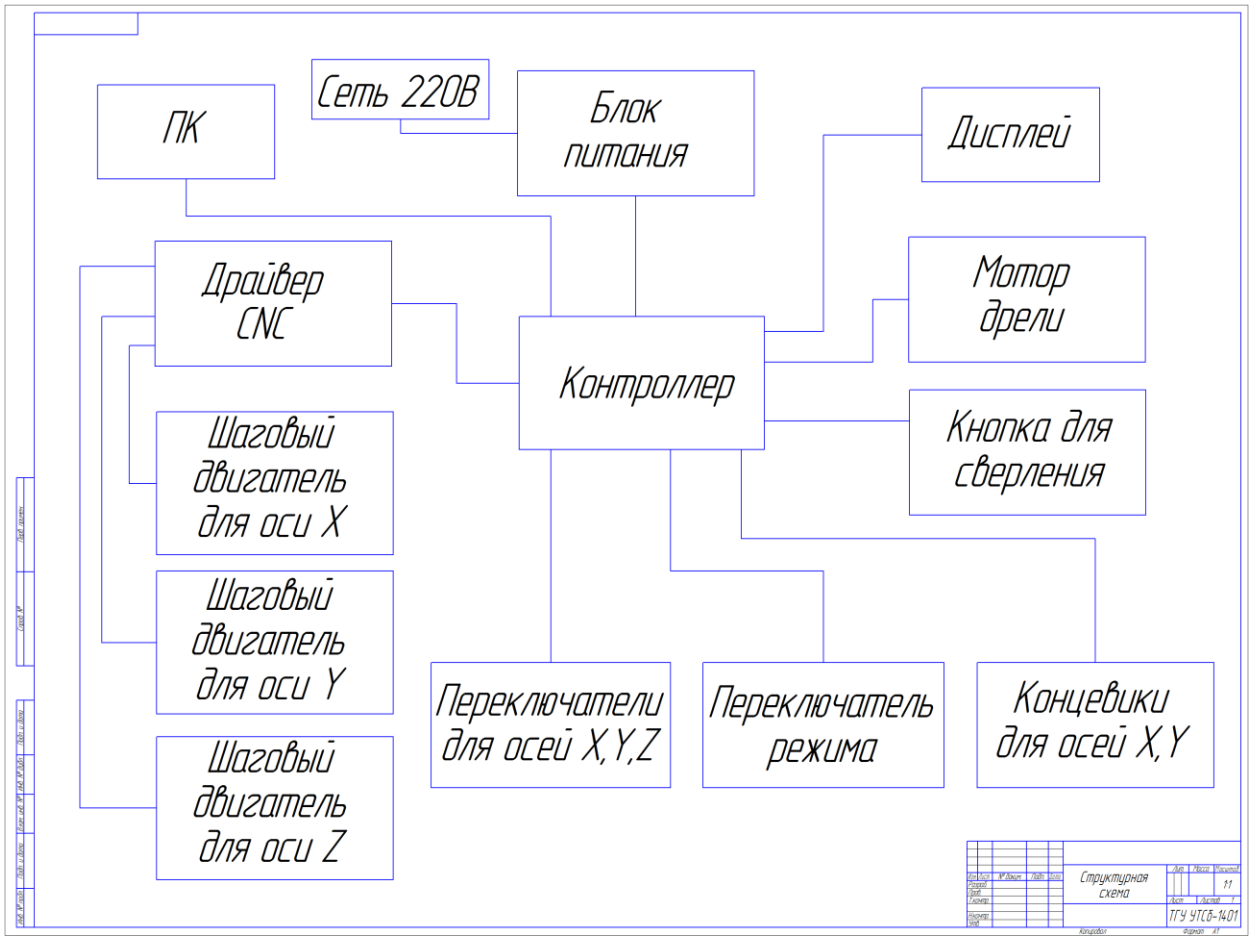

Рисунок 2.1 – Структурная схема автоматизированного сверлильного станка

Питание реализовано от сети 220 Вольт. В блоке питания сетевое напряжение переменного тока преобразуется в постоянный ток более низкого напряжения и подается на контроллер. Контроллер соединён с драйвером CNC Shield V3.0 для работы с шаговыми двигателями для осей X, Y и Z. Для управления шаговыми двигателями в ручном режиме потребуются переключатели, подключенные непосредственно к контроллеру. Также потребуется переключатель режима работы, который будет напрямую подсоединен к контроллеру. Для управления сверлильным станком с

помощью G-кодов в автоматическом режиме необходимо подключить персональный компьютер к контроллеру с помощью USB-интерфейса. Для регулирования перемещения рабочего стола в координатах X и Y используются концевые выключатели. Включение двигателя для шпинделя производится с помощью кнопки. Данные о текущем режиме работы, координаты нахождения шпинделя над рабочим столом обрабатываются контроллером и отображаются на дисплее.

3. Схема электрических соединений

На рисунке 3.1 представлена схема электрических соединений автоматизированного сверлильного станка

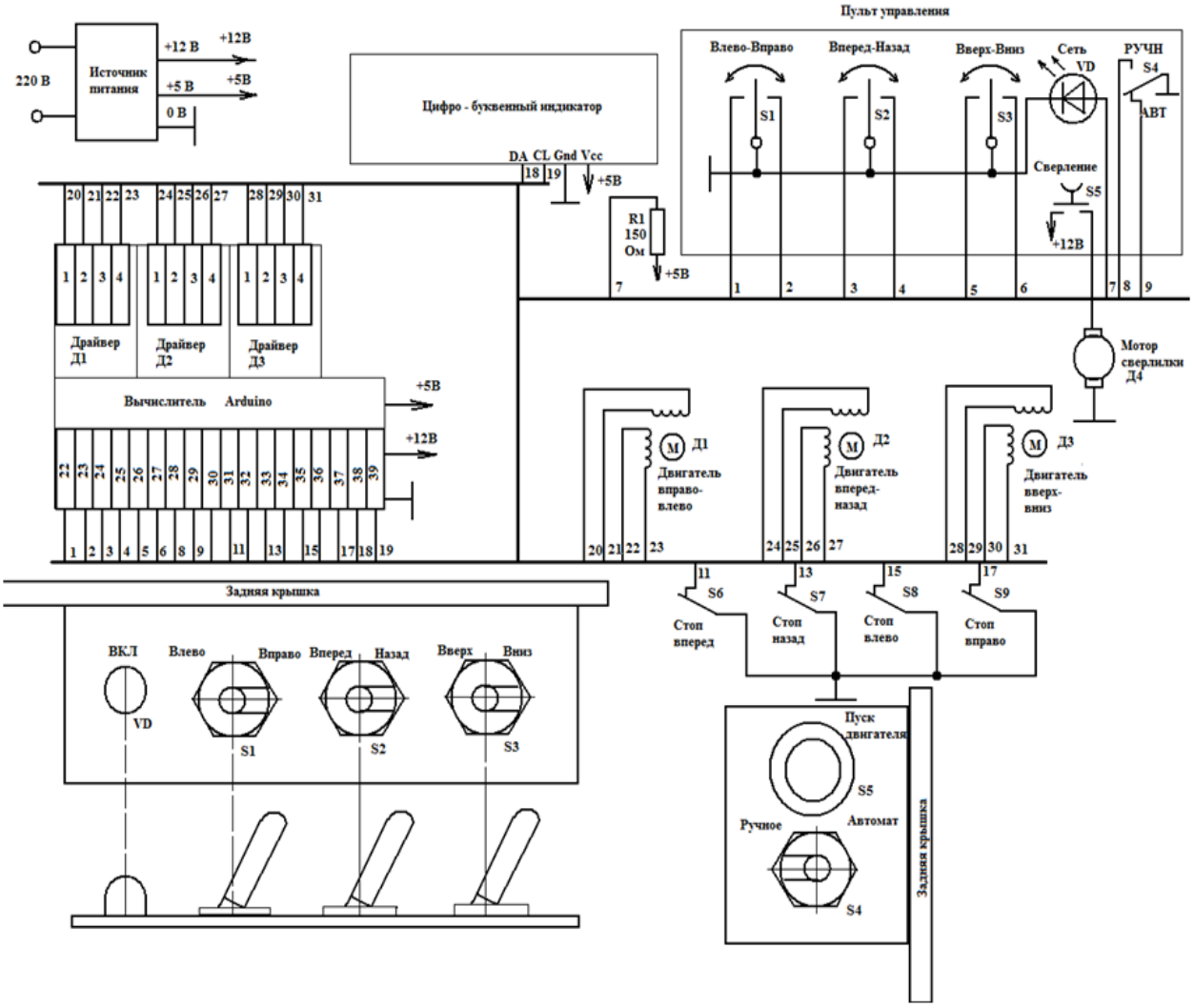

Рисунок 3.1 – Схема электрических соединений автоматизированного сверлильного станка

4. Разработка алгоритма работы программы контроллера в ручном режиме

4.1 Блок-схема алгоритма работы программы контроллера в ручном режиме

Для создания управляющей программы потребуется блок-схема алгоритма работы программы в ручном режиме, представленная на рисунке 4.1.1.

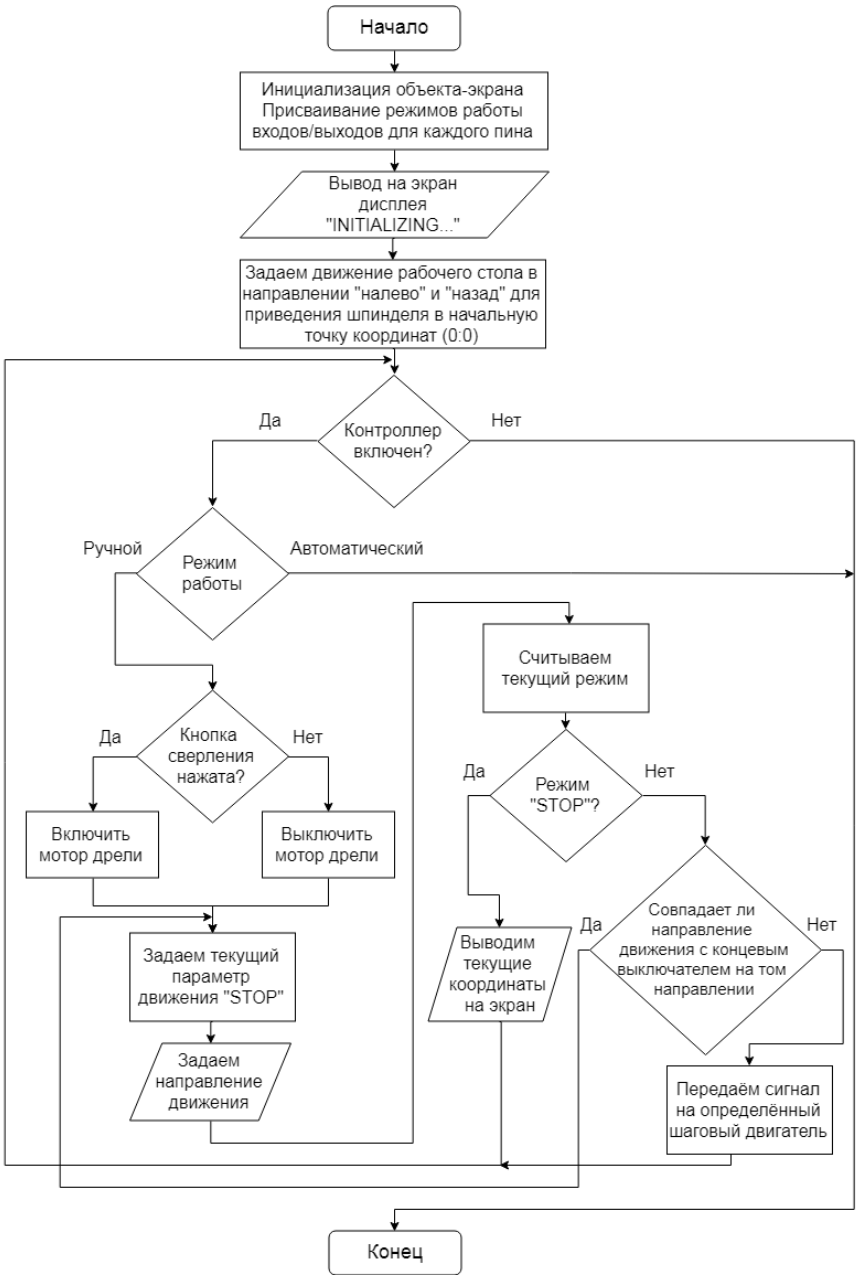

Рисунок 4.1.1 – Блок-схема алгоритма работы программы в ручном режиме

После подключения сверлильного станка к сети 220В, работа должна происходить по следующему алгоритму с определенными условиями:

- Оператор станка включает контроллер;
- Рабочий стол сверлильного станка автоматически перемещается в направлении “налево” и “назад” для приведения шпинделя в начальную точку координат (0;0);
- LCD дисплей должен инициализироваться и отобразить текст “INITIALIZING…”;
- Переключатель режима работы должен быть установлен в положении “ручной режим работы”;
- С помощью трёхпозиционного переключателя “влево-вправо” оператор станка управляет положением рабочего стола по оси X;
- С помощью трёхпозиционного переключателя “вперед-назад” оператор станка управляет положением рабочего стола по оси Y;
- Контроллер должен обрабатывать сигналы концевых выключателей и останавливать работу шаговых двигателей при срабатывании любого из них;
- При остановке работы шаговых двигателей для осей X, Y на LCD дисплее должна отображаться текущая координата расположения сверла над рабочим столом;
- С помощью кнопки “Двигатель шпинделя” оператор станка управляет мотором дрели;
- Трёхпозиционный переключатель “вверх-вниз” используется для управления положением шпинделя по оси Z с помощью шагового двигателя;
- После завершения работы оператор станка выключает контроллер.

4.2 Создание управляющей программы для ручного режима работы

Для программирования используется интегрированная среда разработки Arduino – Arduino IDE.

Для вывода информации будет использоваться четырёхстрочный дисплей с модулем I2C. Это значит, что необходимо установить библиотеку LiquidCrystal_I2C, которая повторяет множество функций стандартной библиотеки LiquidCrystal. Также необходимо подключить стандартную библиотеку Wire, для взаимодействия с I2C/TWI устройствами. Далее нужно объявить объект библиотеки с параметрами дисплея: адрес шины I2C, количество столбцов, количество строк. Это отображено на рисунке 4.2.1.

```
#include <LiquidCrystal_I2C.h>          //Подключение библиотеки для работы с LCD дисплеем по шине I2C
#include <Wire.h>                       //Подключение библиотеки для работы с шиной I2C
LiquidCrystal_I2C lcd(0x3F, 20, 4);     //Объявление объекта библиотеки с указанием параметров дисплея:
                                        //адрес I2C - 0x3F, количество столбцов - 20, количество строк - 4
```

Рисунок 4.2.1 – Фрагмент кода программы с подключением библиотек

Теперь в коде программы работа с дисплеем осуществляется через объект *lcd*.

Далее нужно объявить перечисление с помощью ключевого слова *enum*, который будет состоять из набора именованных констант. Перечисление необходимо для упрощенной установки режима работы для заданных входов/выходов. Каждое состояние переключателя соответствует константе в перечислении, а также пину на контроллере. Например, для переключателя “влево-вправо” в перечислении положение “Налево” соответствует константе *BUTTON_LEFT*, а положение “Вправо” соответствует *BUTTON_RIGHT*. На рисунке 4.2.2 показаны все элементы перечисления, а на рисунке 4.2.3, соответствующие им пины.

```
enum                            //Объявление перечисления
{                               //Начало перечисления
  ENABLE,                       //Работа контроллера
  X_DIR,                        //Направление вращения по оси X
  Y_DIR,                        //Направление вращения по оси Y
  Z_DIR,                        //Направление вращения по оси Z
  X_STEP,                       //Шаг двигателя по оси X
  Y_STEP,                       //Шаг двигателя по оси Y
  Z_STEP,                       //Шаг двигателя по оси Z

  BUTTON_LEFT,                  //Сигнал переключателя "Налево"
  BUTTON_RIGHT,                 //Сигнал переключателя "Направо"
  BUTTON_BACKWARD,              //Сигнал переключателя "Назад"
  BUTTON_FORWARD,               //Сигнал переключателя "Вперёд"
  BUTTON_DOWN,                  //Сигнал переключателя "Вниз"
  BUTTON_UP,                    //Сигнал переключателя "Вверх"
  BUTTON_MANUAL,                //Сигнал переключателя "Ручной режим"
  BUTTON_AUTO,                  //Сигнал переключателя "Автоматический режим"

  LIMIT_SWITCH_LEFT,            //Состояние левого концевого выключателя
  LIMIT_SWITCH_RIGHT,           //Состояние правого концевого выключателя
  LIMIT_SWITCH_BACKWARD,        //Состояние заднего концевого выключателя
  LIMIT_SWITCH_FORWARD          //Состояние переднего концевого выключателя
};                              //Конец перечисления
```

Рисунок 4.2.2 – Фрагмент кода программы с перечислением именованных констант

```
const int pins[] = {
  8, 5, 6, 7, 2, 3, 4, 22, 23, 25, 24, 27, 26, 28, 29, 35, 37, 33, 31  //номера пинов
};
```

Рисунок 4.2.3 – Фрагмент кода программы с константным массивом переменных типа int с номерами соответствующих пинов

На рисунке 4.2.4 показаны переменные для отслеживания текущих координат шпинделя над рабочим столом, а также служебная переменная для обновления времени отображения на дисплее.

```
unsigned long displayUpdateTimestamp = 0;      //Переменная для обновления времени отображения на дисплее

int machineToolX = 0, machineToolY = 0;        //Переменные, обозначающие текущее положение шпинделя
                                               //над рабочим столом по осям координат X и Y
int deltaX = 0, deltaY = 0;                    //Переменные для изменения отображаемых значений шпинделя
                                               //над рабочим столом для осей координат X и Y
```

Рисунок 4.2.4 – Фрагмент кода программы с дополнительными переменными

Также потребуется второе перечисление для текущего режима работы *mode* и предыдущего *oldMode*. Все элементы перечисления режимов работы показаны на рисунке 4.2.5

```
enum                                    //Объявление перечисления
{                                       //Начало перечисления
  STOP,                                 //Остановка движения
  MOVE_LEFT,                            //Движение налево
  MOVE_RIGHT,                           //Движение направо
  MOVE_BACKWARD,                        //Движение назад
  MOVE_FORWARD,                         //Движение вперёд
  MOVE_DOWN,                            //Движение вниз
  MOVE_UP                               //Движение вверх
};                                      //Конец перечисления

int mode = STOP, oldMode = STOP;        //Присваивание текущего режима работы и предыдущего
```

Рисунок 4.2.5 – Фрагмент кода программы с перечислением режимов работы и присваиванием текущего и предыдущего режимов работы

После объявления всех переменных, перечислений и массивов создаются функции, которые будут использоваться при инициализации основной программы *void setup()* и при её выполнении *void loop()*.

Для обновления режима работы создаётся метод UpdateCurrentMode. Он используется для определения и изменения текущего режима работы, а также для изменения переменной, отображающей текущее положение координат шпинделя над рабочим столом на осях X и Y. Весь метод будет состоять из 6 различных условий и двух начальных определений. Вначале нужно установить значение режима *STOP* для того, чтобы при отключенном состоянии всех переключателей шаговые двигатели не продолжали работать в предыдущем режиме работы, а останавливались. Также каждый раз при вызове метода значения *deltaX* и *deltaY* должны быть равны 0 для того, чтобы

не происходило изменение текущих координат положения шпинделя при остановке. Метод *updateCurrentMode* показан на рисунке 4.2.6.

```
void updateCurrentMode()                                                    //Метод для обновление текущего режима работы
{                                                                           //Начало метода
  mode = STOP;                                                              //Начальный режим - остановка режима
  deltaX = deltaY = 0;                                                      //Присваивание переменных для изменения
                                                                            //отображаемых значений шпинделя нулю
  if (pinData[BUTTON_LEFT] == LOW && pinData[LIMIT_SWITCH_LEFT] == LOW)     //Условие при котором возможно движение налево
  {                                                                         //Начало блока команд при выполнении условия
    mode = MOVE_LEFT;                                                       //Установка текущего режима "Движение налево"
    deltaX = -1;                                                            //Задаём положение координаты X-1 за шаг
  }                                                                         //Конец блока команд при выполнении условия

  if (pinData[BUTTON_RIGHT] == LOW && pinData[LIMIT_SWITCH_RIGHT] == LOW)   //Условие при котором возможно движение направо
  {                                                                         //Начало блока команд при выполнении условия
    mode = MOVE_RIGHT;                                                      //Установка текущего режима "Движение направо"
    deltaX = +1;                                                            //Задаём положение координаты X+1 за шаг
  }                                                                         //Конец блока команд при выполнении условия

  if (pinData[BUTTON_BACKWARD] == LOW && pinData[LIMIT_SWITCH_BACKWARD] == LOW)  //Условие при котором возможно движение назад
  {                                                                         //Начало блока команд при выполнении условия
    mode = MOVE_BACKWARD;                                                   //Установка текущего режима "Движение назад"
    deltaY = -1;                                                            //Задаём положение координаты Y-1 за шаг
  }                                                                         //Конец блока команд при выполнении условия

  if (pinData[BUTTON_FORWARD] == LOW && pinData[LIMIT_SWITCH_FORWARD] == LOW)  //Условие при котором возможно движение прямо
  {                                                                         //Начало блока команд при выполнении условия
    mode = MOVE_FORWARD;                                                    //Установка текущего режима "Движение прямо"
    deltaY = +1;                                                            //Задаём положение координаты Y+1 за шаг
  }                                                                         //Конец блока команд при выполнении условия

  if (pinData[BUTTON_DOWN] == LOW)                                          //Условие при котором возможно движение вниз
    mode = MOVE_DOWN;                                                       //Установка текущего режима "Движение вниз"

  if (pinData[BUTTON_UP] == LOW)                                            //Условие при котором возможно движение вверх
    mode = MOVE_UP;                                                         //Установка текущего режима "Движение вверх"
}                                                                           //Конец метода
```

Рисунок 4.2.6 – Фрагмент кода программы с методом *updateCurrentMode*

Для каждого режима работа есть определенные условия, которые зависят от положения переключателя и сигнала от концевого выключателя. Для движения “вверх-вниз” условия будут проще, потому что концевые выключатели по оси Z отсутствуют. Так как кнопки и концевые выключатели работают в режиме INPUT_PULLUP, то при получении сигнала значение на пинах будет LOW.

Далее представлены условия и блоки команд при выполнении для разных режимов работы:

- Для движения налево переменная BUTTON_LEFT и переменная LIMIT_SWITCH_LEFT должны быть равны значению LOW (переключатель должен быть установлен в положение “налево” и левый концевой выключатель не должен быть задет);
- Для движения направо переменная BUTTON_RIGHT и переменная LIMIT_SWITCH_RIGHT должны быть равны значению LOW (переключатель должен быть установлен в положение “направо” и правый концевой выключатель не должен быть задет);
- Для движения назад переменная BUTTON_BACKWARD и переменная LIMIT_SWITCH_BACKWARD должны быть равны значению LOW (переключатель должен быть установлен в положение “назад” и задний концевой выключатель не должен быть задет);
- Для движения вперёд переменная BUTTON_FORWARD и переменная LIMIT_SWITCH_FORWARD должны быть равны значению LOW (переключатель должен быть установлен в положение “вперёд” и передний концевой выключатель не должен быть задет);
- Для движения вниз переменная BUTTON_DOWN должна быть равна значению LOW (переключатель должен быть установлен в положение “вниз”);
- Для движения вверх переменная BUTTON_UP должна быть равна значению LOW (переключатель должен быть установлен в положение “вверх”).

Метод *step* необходим для подачи напряжения на обмотки шаговых двигателей в определенной последовательности и при заданных условиях. Также в нём должно происходить изменение отображаемого значения расположения шпинделя над рабочим столом по осям X и Y. Сам метод состоит из 3 условий и 2 присваиваний значений.

На рисунке 4.2.7 показана структура метода *step*.

```
void step()                                          //Метод step
{                                                    //Начало метода
  if (mode == MOVE_LEFT || mode == MOVE_RIGHT)       //Условие для работы шагового двигателя по оси X
  {                                                  //Начало блока команд при выполнении условия
    digitalWrite(pins[X_STEP], HIGH);                //Подача питания на катушку
    delayMicroseconds(1500);                         //Ожидание 1500 миллисекунд
    digitalWrite(pins[X_STEP], LOW);                 //Отключение питания от катушки
    delayMicroseconds(1500);                         //Ожидание 1500 миллисекунд
  }                                                  //Конец блока команд при выполнении условия

  if (mode == MOVE_BACKWARD || mode == MOVE_FORWARD) //Условие для работы шагового двигателя по оси Y
  {                                                  //Начало блока команд при выполнении условия
    digitalWrite(pins[Y_STEP], HIGH);                //Подача питания на катушку
    delayMicroseconds(1500);                         //Ожидание 1500 миллисекунд
    digitalWrite(pins[Y_STEP], LOW);                 //Отключение питания от катушки
    delayMicroseconds(1500);                         //Ожидание 1500 миллисекунд
  }                                                  //Конец блока команд при выполнении условия

  if (mode == MOVE_DOWN || mode == MOVE_UP)          //Условие для работы шагового двигателя по оси Z
  {                                                  //Начало блока команд при выполнении условия
    digitalWrite(pins[Z_STEP], HIGH);                //Подача питания на катушку
    delayMicroseconds(1500);                         //Ожидание 1500 миллисекунд
    digitalWrite(pins[Z_STEP], LOW);                 //Отключение питания от катушки
    delayMicroseconds(1500);                         //Ожидание 1500 миллисекунд
  }                                                  //Конец блока команд при выполнении условия

  machineToolX += deltaX;                            //Изменение отображаемого значения шпинделя над рабочим столом на оси X
  machineToolY += deltaY;                            //Изменение отображаемого значения шпинделя над рабочим столом на оси Y
}                                                    //Конец метода
```

Рисунок 4.2.7 – Фрагмент кода программы с методом *step*

Для каждой оси координат условие должно включать в себя 2 режима работы. Поэтому используется логический оператор “ИЛИ”, в котором указывается определенный и противоположный ему режимы работы. Для работы шагового двигателя по оси X в условии нужно указать, что блок команд, будет выполняться только при текущем режиме работы соответствующему MOVE_LEFT или MOVE_RIGHT. При выполнении условия на катушку шагового двигателя по оси X подается напряжение, происходит задержка 1500 миллисекунд, отключается питание и снова выполняется задержка. Ожидание в 1500 миллисекунд необходимо для полного выполнения шага двигателя.

Аналогично программируются условия для работы шаговых двигателей по осям Y и Z.

В конце метода осуществляется изменение отображаемого значения шпинделя над рабочим столом с помощью оператора “+=”. Выражение

machineToolX += *deltaX* будет эквивалентно выражению *machineToolX* = *machineToolX* + *deltaX*.

Следующий метод *updateDirectionPin* отображен на рисунке 4.2.8, и он служит для обновления направления вращения шагового двигателя по разным осям координат.

```
void updateDirectionPin()                //Метод updateDirectionPin
{                                        //Начало метода
  boolean xdir = mode == MOVE_LEFT;      //Присваивание направления вращения по оси X в зависимости от режима
  digitalWrite(pins[X_DIR], xdir);       //Назначение направления вращения по оси X

  boolean ydir = mode == MOVE_BACKWARD; //Присваивание направления вращения по оси Y в зависимости от режима
  digitalWrite(pins[Y_DIR], ydir);       //Назначение направления вращения по оси Y

  boolean zdir = mode == MOVE_DOWN;      //Присваивание направления вращения по оси Z в зависимости от режима
  digitalWrite(pins[Z_DIR], zdir);       //Назначение направления вращения по оси Z

  delay(50);                             //Ожидание 50 миллисекунд
}                                        //Конец метода
```

Рисунок 4.2.8 – Фрагмент кода программы с методом *updateDirectionPin*

Каждый раз при вызове метода переменные X_DIR, Y_DIR и Z_DIR будут изменяться в зависимости от текущего режима работы. Переменные xdir, ydir и zdir присваиваются результату условия равенства текущего режима, указанному. Опытным путем было обнаружено, что при значении xdir = TRUE, шаговый двигатель перемещает рабочий стол влево. Значит в сравнении режимов нужно установить истинное значение: *mode* == *MOVE_LEFT*.

Аналогично нужно обновить направление вращений по оси Y и Z, и также установить задержку 50 миллисекунд для корректной работы программы.

Метод *doAction* необходим для начальной инициализации рабочего стола. Так как состояние, при котором сверлильный станок был выключен неизвестно, требуется метод, который будет приводить рабочий стол в начало координат, но при этом реагируя на концевые выключатели в любом месте рабочего пространства.

На рисунке 4.2.9 изображен фрагмент кода программы со структурой метода doAction.

```
void doAction(int action)                                                    //Метод doAction с параметром
{                                                                            //Начало метода
  mode = action;                                                             //Установка текущего режима равным параметру
  updateDirectionPin();                                                      //Вызов метода updateDirectionPin

  while (digitalRead(pins[LIMIT_SWITCH_LEFT + mode - MOVE_LEFT]) == LOW)     //Цикл с условием работы до срабатывания
                                                                             //концевого выключателя
    step();                                                                  //Вызов метода step
}                                                                            //Конец метода
```

Рисунок 4.2.9 – Фрагмент кода программы с методом *doAction*

Сначала метод должен принять параметр, который будет текущим режимом работы для контроллера. После этого, в зависимости от параметра, устанавливается направление вращение шаговых двигателей в методе *updateDirectionPin*. Далее должен выполняться обычный ход программы с помощью метода *step*, но так как инициализация происходит автоматически – требуется задать границы с помощью концевых выключателей. Для этого используется цикл *while*. В условии работы цикла будет считываться показания концевых выключателей в зависимости от текущего режима работы. Определение концевого выключателя осуществляется с помощью выражения *LIMIT_SWITCH_LEFT + mode – MOVE_LEFT*.

Далее потребуется метод для вывода на дисплей текущего положения шпинделя над рабочим столом и данных о текущем режиме работы. Для этого используется метод *displayData*. Так как необходимо обновлять данные о положении шпинделя, то вначале метода следует сделать очистку дисплея от старых данных и перемещение курсора в указанное положение. Для этого в стандартной библиотеке *LiquidCrystal* уже есть функции *clear()* и *setCursor()*. В результате выполнение этих функций экран очищается от старых данных и перемещает курсор в левый верхий угол. Вывод текущего положения шпинделя над рабочим столом будет осуществляться только при остановке работы шаговых двигателей. При движении на экран должно

выводиться сообщение о текущем режиме (инициализации или движении). Для этого использовалось условие с проверкой текущего режима работы, которое показано на рисунке 4.2.10. При выполнении условия на экран выводится текст с координатами шпинделя с помощью функции *print*. Иначе выводится сообщение о том, что происходит движение и требуется дождаться окончания.

```
void displayData()                  //Метод displayData
{                                   //Начало метода
  lcd.clear();                      //Функция для очищения дисплея
  lcd.setCursor(0, 0);              //Позиционирование курсора на первую строку

  if (mode == STOP)                 //Условие при режиме остановки движения
  {                                 //Начало блока команд при выполнении условия
    lcd.print("X=");                //Вывод текста на дисплей
    lcd.print(machineToolX);        //Вывод переменной отображаемого положения на оси X
    lcd.print(", Y=");              //Вывод текста на дисплей
    lcd.print(machineToolY);        //Вывод переменной отображаемого положения на оси Y
  }                                 //Конец блока команд при выполнении условия
  else                              //Иначе
  {                                 //Начало блока команд при невыполнении условия
    lcd.print("MOVING...");         //Вывод текста на дисплей
    lcd.setCursor(0, 1);            //Позиционирование курсора на вторую строку
    lcd.print("PLEASE WAIT...");    //Вывод текста на дисплей
  }                                 //Конец блока команд при невыполнении условия
}                                   //Конец метода
```

Рисунок 4.2.10 – Фрагмент кода программы с методом *displayData*

После написания всех требуемых для ручного режима методов, можно приступить к заполнению основной программы. Следует начать с функции *setup()*. Эта функция запускается один раз при старте работы контроллера. Она нужна для инициализации переменных, определения режимов работы выводов и для перемещения рабочего стола в начальное положение. Фрагмент кода программы с этой функцией отображён на рисунке 4.2.11.

Вначале следует объявить служебную переменную i, которая потребуется для циклов присваивания режимов работы. Далее происходит инициализация дисплея и включение подсветки. После этого идут два цикла,

в которых значениям в перечислении присваиваются номера соответствующих им пинов. Таким образом, первый цикл устанавливает для элементов перечисления режим работы OUTPUT и присваивает им следующие номера пинов:

- ENABLE – 8;
- X_DIR – 5;
- Y_DIR – 6;
- Z_DIR – 7;
- X_STEP – 2;
- Y_STEP – 3;
- Z_STEP – 4;

```
void setup()                                                    //Функция setup
{                                                               //Начало функции
  int i;                                                        //Служебная переменная i

  lcd.begin();                                                  //Инициализации работы с дисплеем
  lcd.backlight();                                              //Включение подсветки LCD дисплея

  for (i = ENABLE; i <= Z_STEP; ++i)                            //Цикл для присваивания режима работы
    pinMode(pins[i], OUTPUT);                                   //значениям до Z_STEP OUTPUT

  for (i = BUTTON_LEFT; i <= LIMIT_SWITCH_FORWARD; ++i)         //Цикл для присваивания режима работы кнопок
    pinMode(pins[i], INPUT_PULLUP);                             //и концевых выключателей INPUT_PULLUP

  digitalWrite(pins[ENABLE], LOW);                              //Подача значения LOW на пин ENABLE

  lcd.print("INITIALIZING...");                                 //Вывод текста
  doAction(MOVE_LEFT);                                          //Метод doAction с параметром MOVE_LEFT
  doAction(MOVE_BACKWARD);                                      //Метод doAction с параметром MOVE_BACKWARD

  Serial.begin(9600);                                           //Открытие последовательного порта, скорость 9600 бит/с
}                                                               //Конец функции
```

Рисунок 4.2.11 – Фрагмент кода программы с функцией *setup()*

Второй цикл устанавливает оставшимся элементам режим работы INPUT_PULLUP и присваивает им следующие пины:

- BUTTON_LEFT – 22;

- BUTTON_RIGHT – 23;
- BUTTON_BACKWARD – 25;
- BUTTON_FORWARD – 24;
- BUTTON_DOWN – 27;
- BUTTON_UP – 26;
- BUTTON_MANUAL – 28;
- BUTTON_AUTO – 29;
- LIMIT_SWITCH_LEFT – 35;
- LIMIT_SWITCH_RIGHT – 37;
- LIMIT_SWITCH_BACKWARD – 33;
- LIMIT_SWITCH_FORWARD – 31;

После этого включается драйвер CNC Shield V3.0 подачей значения *LOW* на пин *ENABLE*.

Далее проводится инициализация с помощью трёх функций: на дисплей выводится текст “INITIALIZING…” с помощью функции *print()*, затем с помощью метода doAction с параметром MOVE_LEFT рабочий стол двигается налево до упора и в конце метод doAction с параметром MOVE_BACKWARD двигает рабочий стол до упора назад. В итоге рабочий стол перемещается в положение, в котором шпиндель над ним находится в координате 0:0.

После того как контроллер запущен, все переменные инициализированы, режимы работы выводов определены и рабочий стол перемещён в начальное положение нужно описать работу контроллера в функции *loop()*. Эта функция бесконечно зациклена до того момента, как контроллер не будет лишён питания или не произойдёт сброс платы Arduino. Она предназначена для активного управления платой Arduino. Функция *loop()* показана на рисунке 4.2.12.

Вначале нужно объявить служебную переменную для цикла считывания данных. Цикл считывания данных нужен для того, чтобы при каждом обновлении в функции *loop*, на контроллер приходили обновлённые данные о состоянии переключателей и концевых выключателей. Обновлённые данные заменяются новыми значениями в массиве pinData. После этого происходит обновление текущего режима работы с помощью метода *updateCurrentMode*.

```
void loop()                                          //Функция loop
{                                                    //Начало функции
  int i;                                             //Объявление служебной переменной i

  for (i = BUTTON_LEFT; i <= LIMIT_SWITCH_FORWARD; ++i)   //Цикл для считывания данных переключателей
    pinData[i] = digitalRead(pins[i]);               //и концевых выключателей

  updateCurrentMode();                               //Метод updateCurrentMode

  if (oldMode != mode)                               //Условие при смене режима работа
  {                                                  //Начало блока команд при выполнении условия
    oldMode = mode;                                  //Присваивание предыдущего режима работы текущему
    displayData();                                   //Метод displayData
    updateDirectionPin();                            //Метод updateDirectionPin
  }                                                  //Конец блока команд при выполнении условия

  if (mode == STOP)                                  //Условие при режиме остановки движения
  {                                                  //Начало блока команд при выполнении условия
    unsigned long time = millis();                   //Объявление переменной time

    if (time - displayUpdateTimestamp >= 1000)       //Условие для обновления значений на дисплее каждую секунду
    {                                                //Начало блока команд при выполнении условия
      displayUpdateTimestamp = time;                 //Присваивание переменной displayUpdateTimestamp текущее время
      displayData();                                 //Метод displayData
    }                                                //Конец блока команд при выполнении условия

    if (Serial.available() > 0)                      //Условие при котором есть доступные данные
      readTargetCoords();                            //Метод readTargetCoords
  }                                                  //Конец блока команд при выполнении условия
  else                                               //Иначе
    step();                                          //Метод step
}                                                    //Конец функции
```

Рисунок 4.2.12 – Фрагмент кода программы с функцией *loop()*

Далее нужно определить, что должно происходить при смене режима работы или остановке движения.

Условие *if (oldMode !=mode)* проверяет текущий режим работы и срабатывает только если метод *updateCurrentMode* обновил его. При срабатывании предыдущий режим работы присваивается значению текущего режима, обновляет данные на дисплее с помощью метода *displayData* и назначает сторону вращения шаговых двигателей в соответствии с режимом работы методом *updateDirectionPin*.

Условие *if (mode == STOP)* срабатывает, если текущий режим работы равен остановке движения. Для ручного режима управления при остановке движения нужно только задать переменную времени, которая будет равна времени, прошедшему с момента входа в условие и сравнить её с переменной *displayUpdateTimestamp*. Это нужно для того, чтобы выводить данные каждую секунду в режиме остановки. В условии сравнения переменной *displayUpdateTimestamp* с временем нужно присвоить значение переменной текущему времени и вызвать метод *displayData*. Далее при любом другом режиме работы надо вызвать метод *step*, который сам определит с каким шаговым двигателем взаимодействовать.

Теперь сверлильный станок инициализируется и перемещается в начальную точку координат, корректно работает в ручном режиме и выводит различные данные о положении рабочего стола на дисплей.

5. G-код и автоматический режим

5.1 G-код

G-код – это общее названия языка программирования для устройств с числовым программным управлением (ЧПУ). Он был разработан в 1960-х годах и окончательно доработан в феврале 1980 года как стандарт RS274D. G-код используется в качестве базового подмножества языка программирования для программы обработки в ПО управления станком. Он имеет множество дополнений, которые вводятся разработчиками аппаратных устройств систем числового программного управления. Программа, написанная с помощью G-кода, состоит из кадров в которых содержатся наборы команд управления. Обычно сначала идут подготовительные команды, затем команды для управления перемещением, после команды выбора режимов обработки материалов и в конце – технологические команды. Комментарии для программы размещаются в круглых скобках и могут располагаться как в отдельной строке, так и после программных кодов. Каждая управляющая команда должна иметь один или несколько параметров, написанные буквами латинского алфавита и параметр, написанный числом. Основные команды языка начинаются с буквы *G*.

В автоматическом режиме сверлильного станка не будут использоваться большое количество параметров для управляющих команд. В основном будет использоваться линейная интерполяция, которая в G-кодах обозначается как G01. Код G01 – это команда для линейной интерполяции, обеспечивающее перемещение по прямой линии. Для передачи команд контроллеру Arduino будет использоваться только код G01 с координатами X и Y.

5.2 Блок-схема алгоритма работы программы контроллера в автоматическом режиме

Для создания управляющей программы потребуется блок-схема алгоритма работы программы в автоматическом режиме, представленная на рисунке 5.2.1.

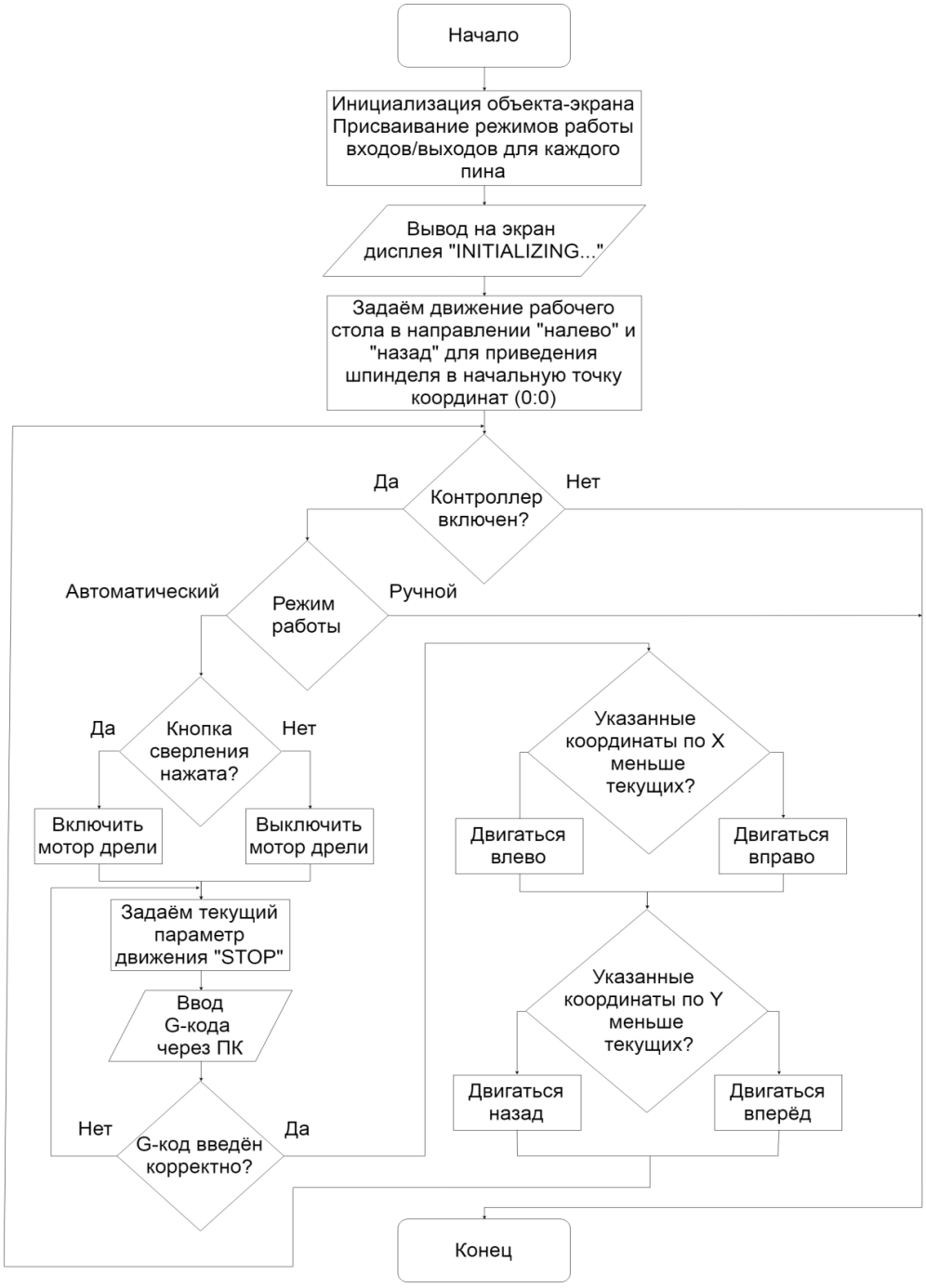

Рисунок 5.2.1 – Блок-схема алгоритма работы программы в автоматическом режиме

В автоматическом режиме алгоритм работы будет похож на алгоритм работы в ручном режиме, но после определения параметра “STOP” изменится определение режима работы. Теперь оно будет определяться программой автоматически, исходя из исходных данных, переданных через ПК.

Работа будет происходить по следующему алгоритму:

- Оператор станка включает контроллер;
- Рабочий стол сверлильного станка автоматически перемещается в направлении “налево” и “назад” для приведения шпинделя в начальную точку координат (0;0);
- LCD дисплей должен инициализироваться и отобразить текст “INITIALIZING…”;
- Переключатель режима работы должен быть установлен в положении “автоматический режим работы”;
- Через ПК должен происходить ввод G-кода, который впоследствии передаётся на контроллер;
- Происходит проверка G-кода на корректность;
- При правильно введённом коде сравниваются координаты по осям X и Y;
- Если указанные координаты по оси X меньше текущих, то контроллер определяет работу шаговых двигателей для движения налево, если координаты больше – то для движения направо;
- Если указанные координаты по оси Y меньше текущих, то контроллер определяет работу шаговых двигателей для движения назад, если координаты больше – то для движения вперёд;

- Контроллер должен обрабатывать сигналы концевых выключателей и останавливать работу шаговых двигателей при срабатывании любого из них;
- При остановке работы шаговых двигателей для осей X, Y на LCD дисплее должна отображаться текущая координата расположения сверла над рабочим столом;
- С помощью кнопки “Двигатель шпинделя” оператор станка управляет мотором дрели;
- Трёхпозиционный переключатель “вверх-вниз” используется для управления положением шпинделя по оси Z с помощью шагового двигателя;
- После завершения работы оператор станка выключает контроллер.

Нужно уделить внимание тому, что автоматически сверлильный станок работает только в осях X и Y. Это связано с тем, что для осей X и Y на рабочем столе присутствуют концевые выключатели, которые позволяют определить размеры рабочего пространства на плоскости. Для оси Z такой возможности нет, так как после каждого завершения работы положение шпинделя остаётся неизвестным.

5.3 Создание управляющей программы для автоматического режима работы

Для создания автоматического режима в управляющей программе потребуется добавить несколько методов и обновить код программы в основной функции выполнения *loop()*.

Сначала потребуется метод *moveToTarget*, который будет сравнивать, заданные через G-код, координаты с текущими координатами шпинделя над рабочим столом. И также будет изменять режим работы в зависимости от

результата сравнения. Фрагмент метода *moveToTarget* для оси X показан на рисунке 5.3.1.

```
if (targetCoordX < machineToolX)      //Сравнение указанных координат по оси X с текущими
{                                     //Начало блока команд при выполнении условия
  mode = MOVE_LEFT;                   //Установка режима работы для движения влево
  deltaX = -1;                        //Задаём положение координаты X-1 за шаг
}                                     //Конец блока команд при выполнении условия
else                                  //Иначе
{                                     //Начало блока команд при невыполнении условия
  mode = MOVE_RIGHT;                  //Установка режима работы для движения вправо
  deltaX = +1;                        //Задаём положение координаты X+1 за шаг
}                                     //Конец блока команд при невыполнении условия

updateDirectionPin();                 //Метода updateDirectionPin

do                                    //Цикл "делать"
{                                     //Начало цикла
  oldMode = mode;                     //Присваивание предыдущего режима работы текущему

  if (targetCoordX < machineToolX)    //Сравнение указанных координат по оси X с текущими
    mode = MOVE_LEFT;                 //Установка режима работы для движения влево
  else                                //Иначе
    mode = MOVE_RIGHT;                //Установка режима работы для движения вправо

  step();                             //Метод step
}                                     //Конец цикла
while (oldMode == mode);              //Пока предыдущий режима работы равен текущему
```

Рисунок 5.3.1 – Фрагмент кода программы с методом *moveToTarget* для оси X

Метод *moveToTarget* состоит из двух частей, каждая из которых изменяет режим работы шаговых двигателей для соответствующей оси. Первая часть метода меняет направление движения для оси X. Далее идёт задержка 300 миллисекунд для того, чтобы рабочий стол закончил все действия связанные с осью X. После этого идёт вторая часть метода, которая меняет направление движения для оси Y.

Для начала в первой части нужно сравнить, указанные через G-код, координаты по оси X с текущими координатами. Так как начало координат было взято слева, то при меньшем значении указанных координат, нужно задать режим работы *MOVE_LEFT* и обозначить, что изменение координат

будет происходить с *deltaX = -1*. При обратном результате, аналогично указать режим работы MOVE_RIGHT и изменение *deltaX = +1*.

После того, как режим работы определён, следует обновить направление вращения шаговых двигателей с помощью метода *updateDirectionPin*.

Далее создаётся цикл, который будет работать, пока режим работы не изменится. То есть пока режим работы не изменится на *STOP*, каждый шаг, предыдущий режим работы будет приравниваться текущему, будет происходить проверка режима работы и вызываться метод *step*.

Для работы предыдущего метода пригодились переменные *targetCoordX* и *targetCoordY*. Для считывания этих переменных создаётся метод *readTargetCoords*, который будет считывать G-код и передавать значения координат по осям X и Y переменным *targetCoordX* и *targetCoordY*. Он показан на рисунке 5.3.2.

```
void readTargetCoords()                                                    //Метод readTargetCoords
{                                                                          //Начало метода
  int i, b;                                                                //Служебные переменные i,b

  for (i = 0; i < BUFFER_SIZE - 1; ++i)                                    //Цикл для считывания всех символов после ввода
  {                                                                        //Начало цикла
    b = Serial.read();                                                     //Считывания ввода

    if (b == -1)                                                           //Условия для принудительного окончания цикла
     break;                                                                //Принудительное окончание цикла

    serialBuffer[i] = b;                                                   //Запись в символов в массив

    delay(10);                                                             //Ожидание 10 миллисекунд
  }                                                                        //Конец цикла

  serialBuffer[i] = '\0';                                                  //Добавление окончания строки

  if (sscanf(serialBuffer, "G01 X%d,Y%d", &targetCoordX, &targetCoordY) == 2) //Условие для считывания 2 переменных
    moveToTarget();                                                        //Метод moveToTarget
}                                                                          //Конец метода
```

Рисунок 5.3.2 – Фрагмент кода программы с методом *readTargetCoords*

Сначала нужно объявить служебные переменные *i* и *b*. С помощью цикла считывается G-код, введённый через компьютер, записывается в переменную типа строка *serialBuffer*. Для корректной работы со строкой в конце добавляется символ “\0”, который будет обозначать конец строки.

После этого создаётся условие, которое считывает G-код в формате “G01 X%d,Y%d”, передавая с помощью указателей значения координат, и срабатывает, только если передаваемых значения будет два.

Теперь, когда методы для работы в автоматическом режиме добавлены, нужно обновить функции *setup()* и *loop()*. В функции setup() во время инициализации нужно ещё добавить стандартную функцию *begin* в классе *Serial*. Она инициализирует последовательное соединение и задаёт скорость передачи данных в бит/с. Для передачи G-кода на контроллер достаточно скорости 9600 бит/с. В итоге получится строка, изображенная на рисунке 5.3.3

```
Serial.begin(9600);     //Открытие последовательного порта, скорость 9600 бит/с
```

Рисунок 5.3.3 – Фрагмент кода программы с открытием последовательного порта

Далее в функции *loop()* нужно добавить условие при котором будет активироваться автоматический режим. Этим условием будет проверка вводимых значений с помощью функции *Serial.available*. На рисунке 5.3.4 показана структура условия и вызова метода *readTargetCoords* при его выполнении.

```
if (Serial.available() > 0)                    //Условие при котором есть доступные данные
  readTargetCoords();                          //Метод readTargetCoords
```

Рисунок 5.3.4 – Фрагмент кода программы с условием для автоматической работы

Теперь на контроллер можно загрузить готовую программу, которая поддерживает работу автоматизированного сверлильного станка в ручном и автоматическом режимах.

6. Экспериментальное тестирование и отладка программы

После написания всех методов и основных функций программы производится её тестирование и отладка.

При включении происходит первая инициализация компонентов, контроллера и дисплея, которая показана на рисунке 6.1

Рисунок 6.1 – Первая инициализация сверлильного станка

На передней панели управления загорается светодиод, информирующий о том, что сверлильный станок подключен к сети. Он изображён на рисунке 6.2.

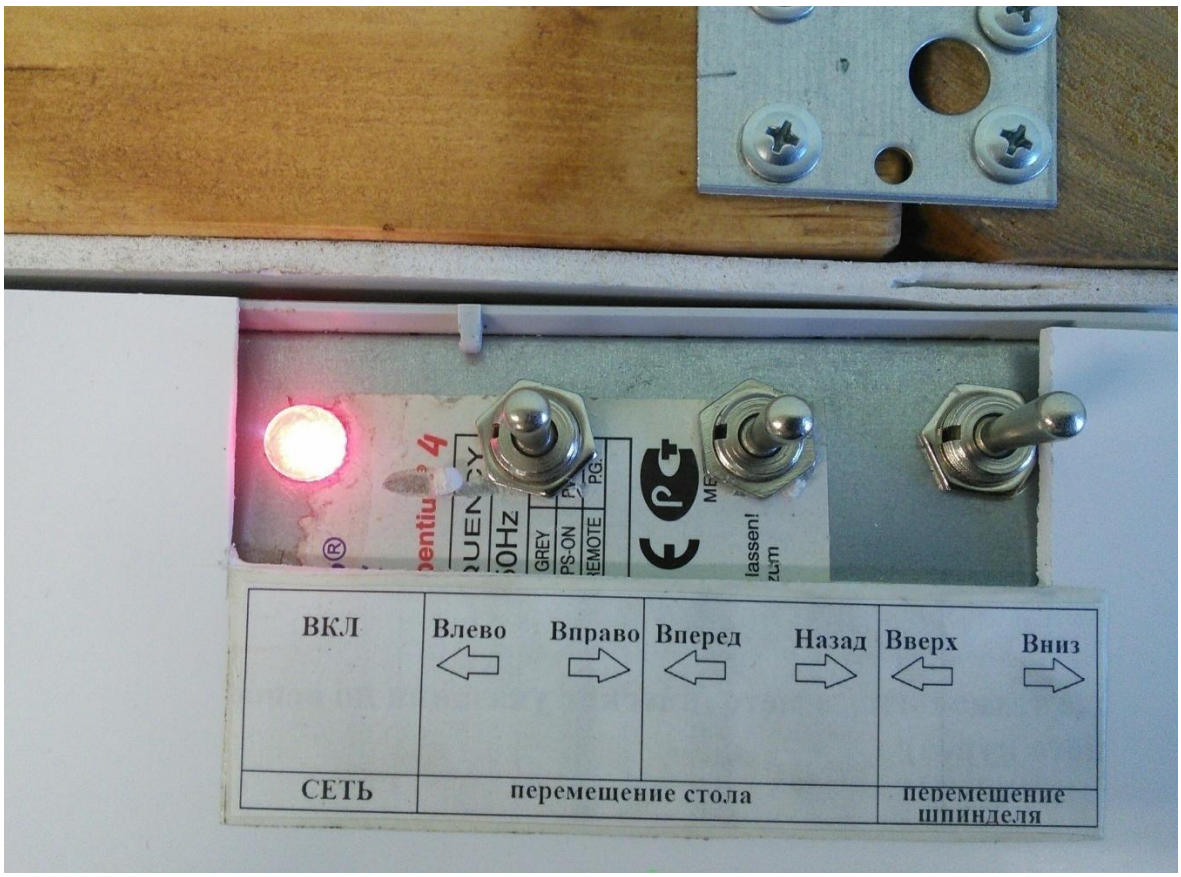

Рисунок 6.2 – Светодиод с отражением состояния работы сверлильного станка

Далее происходит программная инициализация контроллером, рабочий стол двигается в координаты (0:0), на дисплее отображается режим инициализации. На рисунке 6.3 показан текст, который должен отображаться при первой инициализации. На рисунке 6.4 показано положение станка после инициализации и его координаты (0:0).

Рисунок 6.3 – Дисплей в режиме инициализации

Рисунок 6.4 – Положение станка и его координаты в начальном положении

Далее нужно поставить переключатель в положение для ручного режима работы, что изображено на рисунке 6.5.

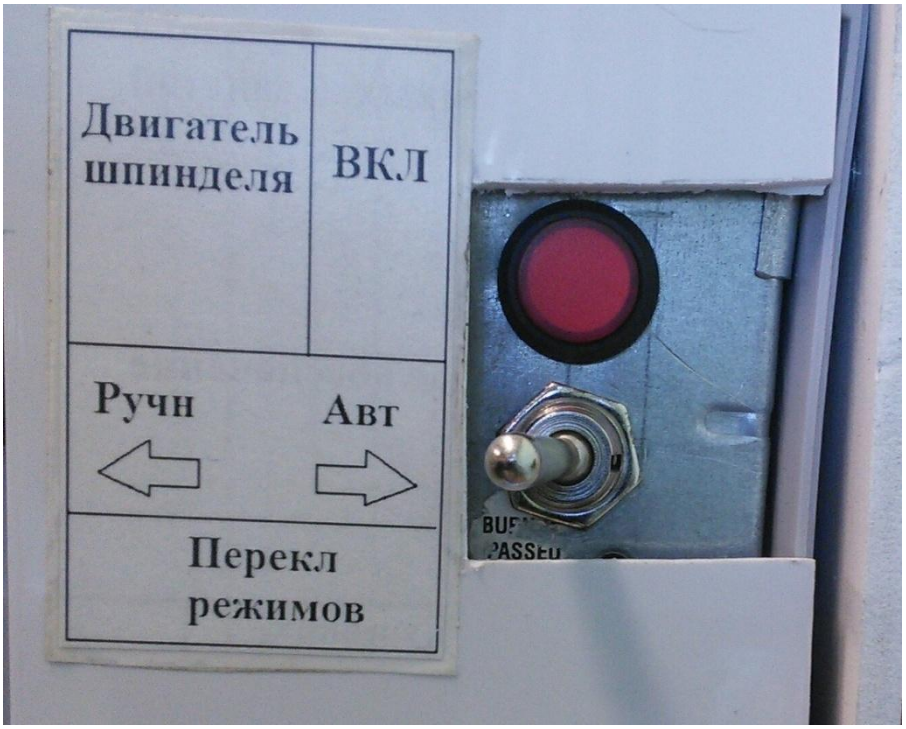

Рисунок 6.5 – Состояние переключателя в ручном режиме

При управлении рабочим столом с помощью переключателей на передней панели на дисплее отображается текст с режимом работы. Это показано на рисунке 6.6.

Рисунок 6.6 – Дисплей в режиме движения рабочего стола или движения шпинделя

После того как переключатели установлены в нейтральное состояние и рабочий стол не движется, программа переходит в режим *STOP* и отображает текущие координаты на дисплее. На рисунке 6.7 отображён режим *STOP*.

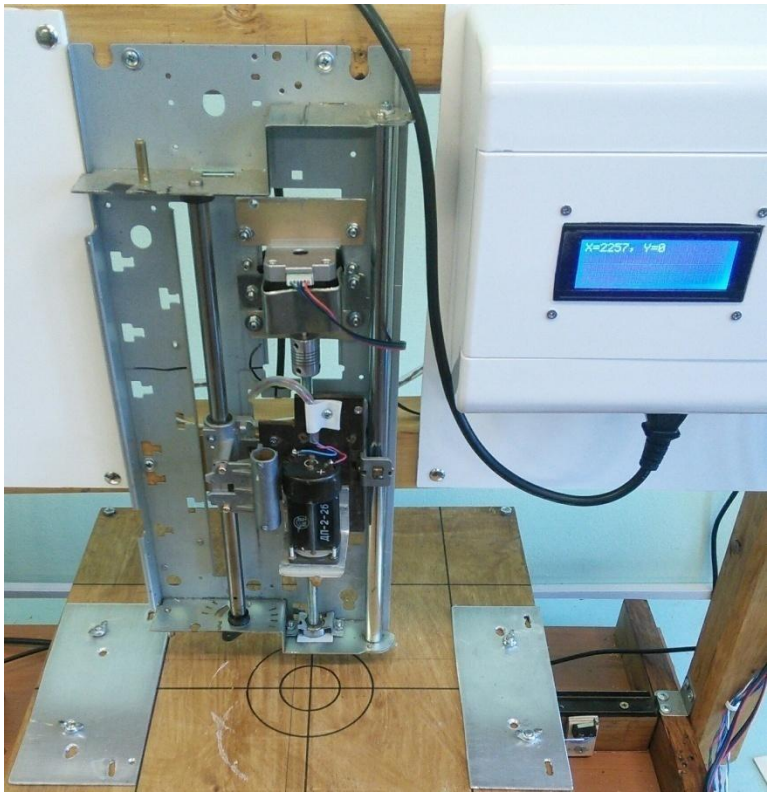

Рисунок 6.7 – Сверлильный станок в режиме *STOP*

Для работы в автоматическом режиме на боковой панели станка нужно установить переключатель в автоматический режим работы. Это показано на рисунке 6.8.

Рисунок 6.8 – Боковая панель с переключателем, установленным в автоматический режим работы

Далее через компьютер нужно передать G-код через монитор порта в Arduino IDE. На рисунке 6.9 показан G-код, который будет передан через ПК на контроллер.

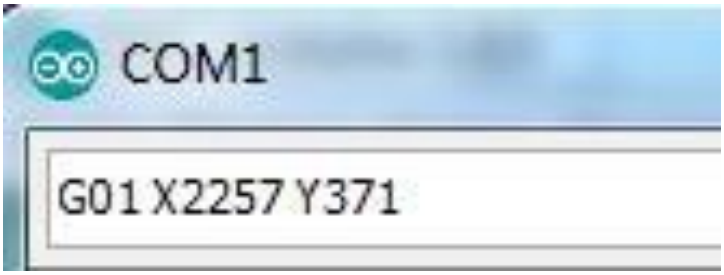

Рисунок 6.9 – G-код, передающийся через монитор порта

После передачи, контроллер автоматически переместит рабочий стол в указанные G-кодом координаты и после остановки отобразит их на дисплее. На рисунке 6.10 показан дисплей сверлильного станка после остановки в указанных координатах.

Рисунок 6.10 – Состояние дисплея после остановки в указанных координатах

Управление движением шпинделя по оси Z осуществляется с помощью переключателя “Вверх-вниз” на передней панели управления. На рисунке 6.11 показано изменение положения шпинделя по оси Z.

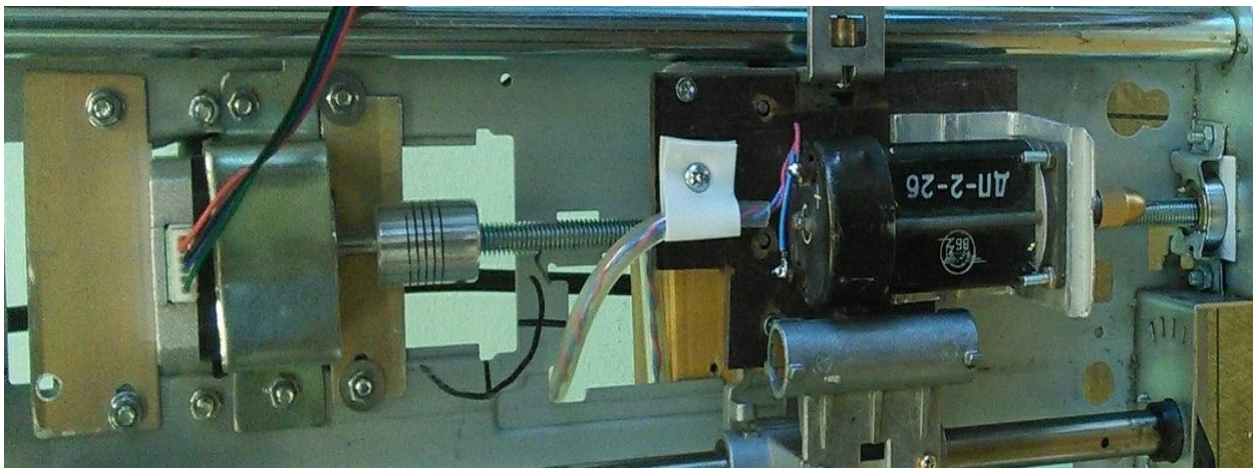

Рисунок 6.11 – Изменение положение шпинделя

В ходе разработки программного обеспечения для автоматизированного рабочего станка были выявлены проблемы с движением рабочего стола за пределами его рабочей зоны, которые впоследствии были решены улучшением алгоритмов работы программы. Также в режиме автоматической работы движение станка может осуществляться с погрешностью 1-2 условных шага.

Проблемы с мерцанием дисплея тоже были решены, посредством изменения его инициализации и ввода дополнительного условия для его ежесекундного обновления.

7. Безопасность и экологичность проекта

Корпус сверлильного станка изготовлен из фанеры и металлических деталей. Во время обработки следует соблюдать инструкции техники безопасности при ручной обработке материалов. Перед работой нужно надеть специальную одежду и защитные очки. Далее следует подготовить рабочее место и проверить исправность всех рабочих инструментов. Для выполнения технологических операций на станке следует надежно закреплять печатные платы с использованием различных тисков, подкладок и других средств, необходимых для поддержания платы в неподвижном состоянии. Также перед эксплуатацией необходимо проверить надежность всех соединений. При обнаружении неисправности провести замену или подключение проводников в соответствии с документацией, в зависимости от условий. После проверки подключения необходимо активировать контроллер с помощью кнопки на панели управления. Далее дождаться инициализации работы системы. По завершению инициализации изменять положение печатной платы только при выключенном моторе дрели и положении переключателей в нейтральном состоянии. При замене каких-либо элементов конструкции необходимо сначала обесточить станок.

Среди элементов автоматизированного сверлильного станка отсутствуют детали, несущие существенный вред окружающей среде. Также сверлильный станок работает от питания сети без использования аккумуляторов.

8. Экономическая эффективность

На рисунке 8.1 изображена таблица с расчётом всех компонентов рабочей системы. В конце таблице показана общая стоимость всех компонентов.

№	Наименование	Кол.	Цена за шт., р.	Цена, р.
1	Контроллер Arduino MEGA 2560	1	2990	2990
2	Блок питания	1	590	590
3	Дисплей с модулем I2C четырёхстрочный	1	790	790
4	Драйвер CNC Shield V 3.0	1	187	187
5	Шаговые двигатели	3	900	2700
6	Привод мотора дрели	1	790	790
7	Трёхпозиционный переключатель	3	25	75
8	Двухпозиционный переключатель	1	20	20
9	Концевой выключатель	4	10	40
10	Кнопка питания мотора дрели	1	15	15
11	Кнопка включения	1	10	10
			Итого	8207 р.

Рисунок 8.1 – Экономический расчёт

Заключение

В данной выпускной бакалаврской работе была показана важность автоматизации производства с помощью внедрения автоматизированных станков. Было написано программное обеспечение для работы с автоматизированным сверлильным станком. Была разработана структурная схема и схема электрических соединений. Также были разработаны блок схемы для ручного и автоматического режимов работы, с помощью которых пошагово показывался алгоритм работы системы. Для работы контроллера была написана программа на языке программирования устройств Arduino, который основан на C/C++. Представлено экспериментальное тестирование и описаны его результаты. Произведена оценка безопасности и экологичности сверлильного станка. Выполнена оценка экономической эффективности, выполнен расчёт всех компонентов системы.

Список используемой литературы

1. Шилдт, Герберт: C++: Базовый курс, 3 - е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2010. – 624 с.: ил.
2. Москатов Е. А. Электронная техника. Начало. – 3-е изд., перераб. и доп. – Таганрог, 204 с., ил
3. Соммер У. Программирование микроконтроллерных плат Arduino/Freeduino: пер с нем. – 2-е изд., перераб. и доп. – СПБ.: БХВ-Петербург, 2016, – 256 с.: ил. – (Электроника)
4. Ревич Ю.В. Занимательная электроника. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2009. — 720 с.: ил.
5. Блум Джереми. Изучаем Arduino: инструменты и методы технического волшебства: Пер. с англ. – СПб.: БВХ-Петербург, 2015. – 336 с.: ил. ISBN 978-5-9775-3585-4
6. Усатенко С.Т., Каченюк Т.К., Терехова М.В. Выполнение электрических схем по ЕСКД: Справочник. – М.: Издательство стандартов, 1989. – 325 с.
7. Кузин А.В, Жаворонков М.А.: Микропроцессорная техника. М.: Издательский центр «Академия», 2007.- 304 с.
8. Ратмиров В.А, Шаговые двигатели для систем автоматического управления – Рипол Классик, 2013.- 134 с.
9. [Электронный ресурс] – Электрон. дан.: Интернет-портал «Амперка», 2018. – Режим доступа: http://amperka.ru/, свободный. – Загл. с экрана.
10. [Электронный ресурс] – Электрон. дан.: Интернет-портал «Arduino», 2018. – Режим доступа: https://www.arduino.cc/, свободный. – Загл. с экрана.
11. [Электронный ресурс] – Электрон. дан.: Интернет-портал «Arduino.ru», 2018. – Режим доступа: http://arduino.ru/, свободный. – Загл. с экрана.

12.[Электронный ресурс] – Электрон. дан.: Интернет-портал «Arduino-diy.com», 2018. – Режим доступа: http://arduino-diy.com/, свободный. – Загл. с экрана.
13.[Электронный ресурс] – Электрон. дан.: Интернет-портал «iarduino», 2018. – Режим доступа: https://wiki.iarduino.ru/, свободный. – Загл. с экрана.
14.[Электронный ресурс] – Электрон. дан. – М.: Интернет-портал «Всё об ардуино», 2018. – Режим доступа: https://arduinomaster.ru/, свободный. – Загл. с экрана.
15.[Электронный ресурс] – Электрон. дан. – М.: Интернет-портал «Хабр», 2018. – Режим доступа: https://habr.com/, свободный. – Загл. с экрана.
16.Alexandru Morar. Five Phase Pentagon Hybrid Stepper Motor Intelligent Half/Full Driver, [Электронный ресурс]: https://doaj.org/
17.Carlos Morón, Daniel Ferrández, Pablo Saiz, Gabriela Vega, Jorge Pablo Díaz. New Prototype of Photovoltaic Solar Tracker Based on Arduino [Электронный ресурс]: https://doaj.org/
18.Gheorghe Baluta. Open-Loop Control of A Bipolar Stepper Motors Using The Specialized Integrated Circuits [Электронный ресурс]: https://doaj.org/
19.Tzu-Heng Hsu, Yen-Cheng Chiang, Wei-Tun Chan, Shih-Jui Chen. A Finger Exoskeleton Robot for Finger Movement Rehabilitation [Электронный ресурс]: https://doaj.org/
20.Alexandru Morar. Study on modelling and simulation of permanent magnet stepping motor by MATLAB/SIMULINK [Электронный ресурс]: https://doaj.org/