Национальный исследовательский университет «МЭИ»

Институт Радиотехники и электроники им. В.А. Котельникова

КУРСОВАЯ РАБОТА

по дисциплине

«Цифровая и микропроцессорная техника»

Группа:	Номер группы
Выполнил:	ФИО
Вариант:	Номер варианта
Проверили:	1-й Преподаватель
	2-й Преподаватель

Москва – год

СОДЕРЖАНИЕ

1 ЗАДАНИЕ

Разработать систему управления шаговым двигателем на базе микроконтроллера PIC18F2520, где управляющие команды (направление и скорость вращения) передаются по UART от внешнего устройства (например, ПК). Для этого спроектировать электрическую схему подключения двигателя с учётом драйвера, реализовать приём данных через UART с использованием прерываний, настроить таймер для точной генерации шаговых импульсов, а также написать и отладить управляющую программу, обеспечивая плавное и адекватное реагирование двигателя на входные команды.

2 АНАЛИЗ ЗАДАНИЯ

Необходимо создать схему, которая состоит из терминала, с помощью которого через UART будут подаваться команды, микроконтроллера PIC18F2520, который обрабатывает команды и контролирует скорость и направление вращения шагового двигателя. Драйвер, в свою очередь, преобразует команды с микроконтроллера в силовые импульсы и приводит шаговый двигатель в движение.

3 ВЫБОР ЭЛЕМЕНТОВ СХЕМЫ

Для разработки схемы был выбран шаговый двигатель NEMA17 и драйверы L297 и L298N. Два драйвера используются в соответствии с даташитом. Схема подключения драйверов к шаговому двигателю показана на рисунке 1.

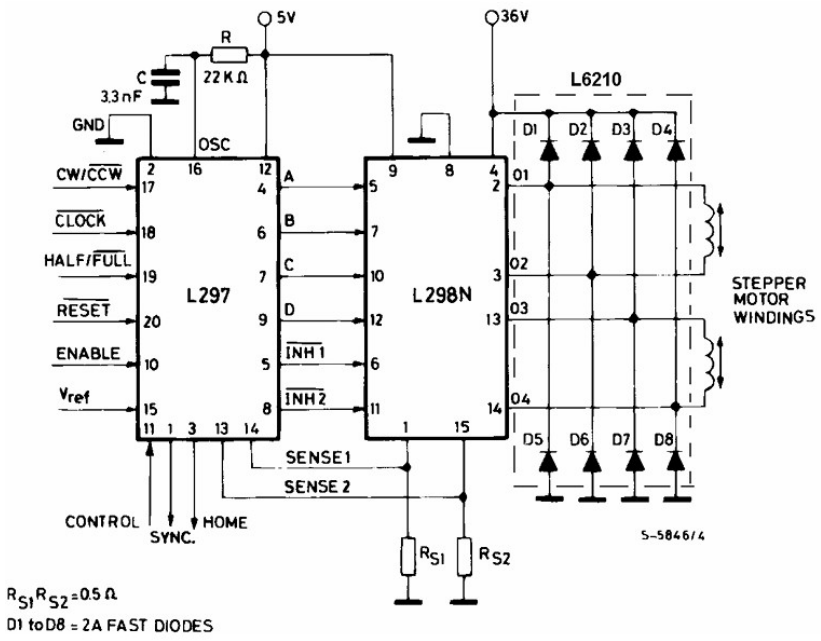

Рисунок 1 – Подключение драйверов к шаговому двигателю

Пины драйвера L297 имеют следующий функционал:

1. Питание и общие выводы

- Vss (пин 20): пин, к которому подключается питание
- GND (пин 10): пин, подключаемый к «земле»
- Vref (пин 12): регулирует ток на обмотках шагового двигателя. На него подается опорное напряжение, которое задает порог срабатывания датчиков тока (SENSE A/B). Изменяя это напряжение, контролируется ток на обмотках двигателя.
- SENSE 1 (пин 1) и SENSE 2 (пин 15): Входы для сигналов с токоизмерительных резисторов (шунтов), подключенных к силовой части. L297 сравнивает падение напряжения на этих резисторах с Vref. При превышении порога активируется схема ШИМ (PWM) для стабилизации тока, предотвращая перегрев двигателя и драйвера.
- OSC (пин 16) – пин, который через внешнюю RC-цепь задаёт частоту внутреннего генератора ШИМ.

2. Входные управляющие сигналы (от микроконтроллера)

- CLOCK (пин 18): Тактовый вход (Step). Каждый положительный фронт (переход из 0 в 1) сигнала на этом пине сдвигает внутренний счетчик и

заставляет двигатель сделать один шаг (в зависимости от выбранного режима). Основной управляющий сигнал.

- CW/CCW (пин 17): Направление вращения. 1 — clockwise (по часовой), 0 — counter-clockwise (против часовой).
- HALF/FULL (пин 19): Выбор режима шага. 1 — полушаговый режим, 0 — полношаговый режим.
- RESET (пин 20): Активный низкий уровень. При подаче 0 сбрасывает внутренний декодер в начальное состояние. В рабочем режиме должен быть 1.
- ENABLE (пин 10): Разрешение работы. 1 — разрешено, микросхема генерирует управляющие сигналы; 0 — все выходы A, B, C, D переходят в высокоимпедансное состояние (Z), мотор обесточивается.

3. Выходные сигналы к силовому драйверу (к L298N)

Эти выводы формируют последовательность коммутации фаз.

- A, B, C, D (пины 13, 14, 16, 3): Основные выходы управления фазами двигателя. Они напрямую подключаются к входам IN1, IN2, IN3, IN4 силового драйвера (например, L298N).
- INH1, INH2 (пины 11 и 4): Сигналы разрешения (Inhibit) для ШИМ. Они используются для отсечки тока через соответствующие плечи силового моста (L298N) во время фазы ШИМ-стабилизации тока. Подключаются к входам EN A и EN B драйвера L298N.

4. Выходы для индикации и контроля

- HOME (пин 4): используется для индикации начального положения двигателя (ABCD = 0101). При таком положении обмоток выдаёт 1.
- SYNC (пин 5): Выход синхронизации ШИМ. На этом выводе появляются импульсы ШИМ (частота задается внешним OSC). Используется для синхронизации нескольких шаговых двигателей.

Для контроля скорости и направления вращения двигателя порты микроконтроллера RB3 и RB4 будут подключаться к пинам драйвера CW/CWW и CLOCK соответственно.

На пины ENABLE и RESET подаём напряжение с источника питания 5 В, чтобы драйвер позволял двигателю работать постоянно. Пин HALF/FULL устанавливаем на землю для реализации полношагового режима. Пины CONTROL, HOME в работе не используются.

Драйвер L298N является силовым модулем, который передаёт импульсы тока на обмотки шагового двигателя. Пины драйвера имеют следующий функционал:

- Выводы SENSE A и SENSE B (пины 1 и 15) — входы/выходы обратной связи по току. Предназначены для подключения низкоомных шунтирующих резисторов (номинал 0,1–0,5 Ом) в цепях силовых выходов. Обеспечивают возможность контроля и программного ограничения тока в нагрузке. В простых схемах могут быть заземлены.
- Силовые выходы канала A — выводы OUT1 и OUT2 (пины 2 и 3). К ним подключается нагрузка: одна обмотка шагового двигателя или один двигатель постоянного тока.
- Вход силового питания — вывод Vs (пин 4). На него подаётся напряжение питания двигателей (до 46 В).
- Управляющие входы канала A — выводы IN1 и IN2 (пины 5 и 7). Логические входы, определяющие направление вращения и режим работы двигателя канала A (вращение вперёд/назад, торможение).
- Вход разрешения канала A — вывод ENA (пин 6). При подаче логической "1" активирует работу канала A. На этот вход может подаваться ШИМ-сигнал для регулировки скорости двигателя.
- Общий вывод (земля) — вывод GND (пин 8). Должен быть соединён с общим проводом всех источников питания (силового, логического и управляющей схемы).

- Вход питания логической части — вывод Vss (пин 9). Требует стабилизированного напряжения +5 В для питания внутренней логики управления микросхемы.
- Управляющие входы канала B — выводы IN3 и IN4 (пины 10 и 12). Функционально аналогичны IN1 и IN2, но управляют вторым силовым каналом.
- Вход разрешения канала B — вывод ENB (пин 11). Выполняет те же функции, что и ENA, но для канала B.
- Силовые выходы канала B — выводы OUT3 и OUT4 (пины 13 и 14). Предназначены для подключения второй обмотки шагового двигателя или дополнительного двигателя постоянного тока.

4 НАСТРОЙКА ТАЙМЕРА ПРЕРЫВАНИЙ

Для реализации прерываний будем использовать внутренний генератор INTIO67. Частоту установим равной 8 МГц, т.к. на этой частоте генератор работает стабильно. Значит, команды будут выполняться с частотой $F_{osc}/4=2$ МГц, т.е. период выполнения будет равен 0,5 мкс. Используем предделитель 1:8, тогда время выполнения одного такта займёт 4 мкс.

5 НАСТРОЙКА UART

Скорость передачи 9600 бод выбрана как оптимальный компромисс между несколькими факторами. Во-первых, эта скорость поддерживается большим количеством терминальных устройств. Во-вторых, при тактовой частоте микроконтроллера 8 МГц данная скорость обеспечивает минимальную погрешность тактирования - расчетное значение делителя составляет 51.08, что при округлении до 51 дает погрешность менее 0.5%, что гарантирует надежную передачу без ошибок синхронизации. На рисунке 2 представлены формулы для расчета скорости обмена данными при разных значениях регистров.

SYNC	BRGH = 0	BRGH = 1
0	(Асинхронный) Скорость обмена = F_{OSC} / (64 (X + 1))	(Асинхронный) Скорость обмена = F_{OSC} / (16 (X + 1))
1	(Синхронный) Скорость обмена = F_{OSC} / (4 (X + 1))	(Синхронный) Скорость обмена = F_{OSC} / (4 (X + 1))

X = значение регистра SPBRG (от 0 до 255)

Рисунок 2 – Формулы расчета скорости

$$\text{скорость обмена} = \frac{F_{\text{осц}}}{16(X+1)}$$

$$X = \frac{F_{\text{осц}} - \text{целевая скорость обмена} \cdot 16}{16 \cdot \text{целевая скорость обмена}} = \frac{8000000 - 9600 \cdot 16}{16 \cdot 9600} = 51.08$$

$$\text{Вычисленное значение скорости обмена} = \frac{8000000}{16(51+1)} = 9615$$

$$\text{Ошибка} = 100 \cdot \frac{\text{Вычисленное} - \text{Желаемое}}{\text{Вычисленное}} = 100 \cdot \frac{9615 - 9600}{9615} = 0.16\%$$

Настройки UART включают асинхронный режим работы, 8 бит данных, 1 стоп-бит и отсутствие контроля четности. Такая конфигурация является наиболее распространенной для задач мониторинга, поскольку обеспечивает достаточную надежность при минимальной избыточности передаваемых данных.

6 РАЗРАБОТКА ПРОГРАММЫ

Блок-схемы алгоритма показаны на рисунках 3, 4, 5, 6.

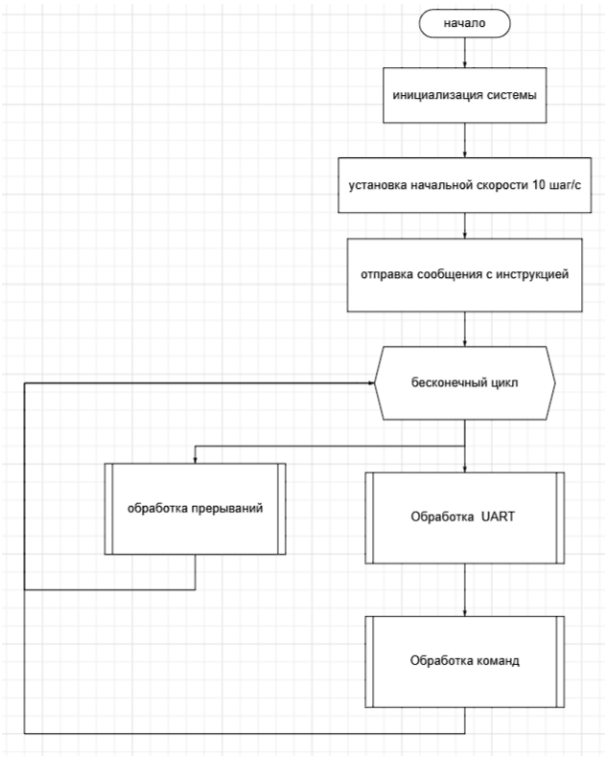

Рисунок 3 – Блок-схема основной программы

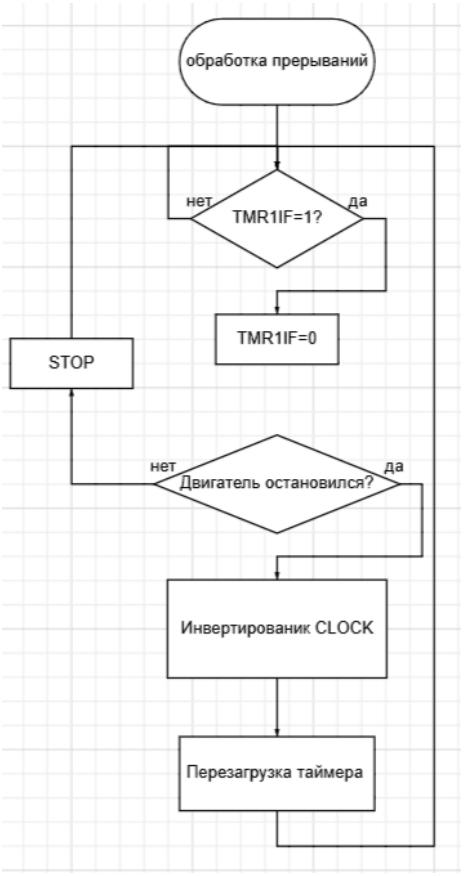

Рисунок 4 – Блок-схема обработки прерываний

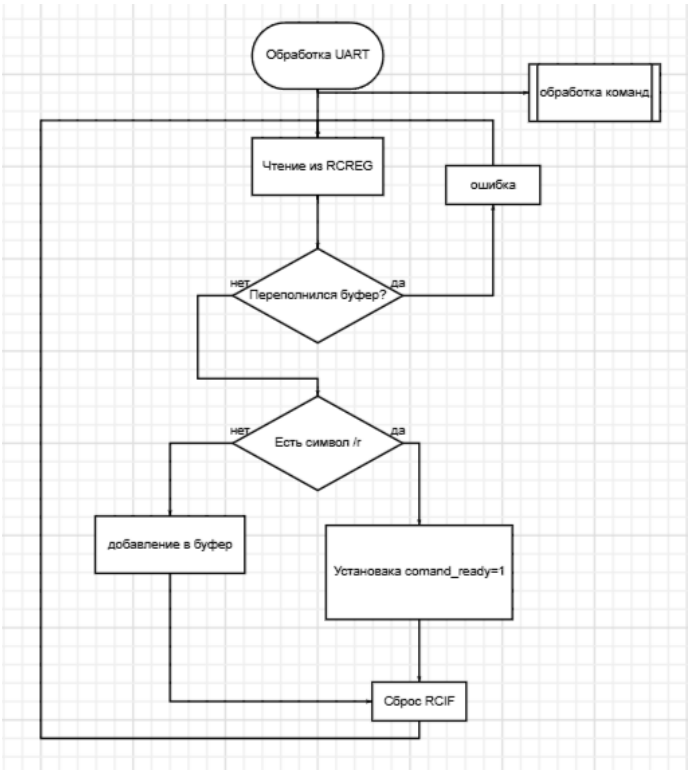

Рисунок 5 – блок-схема обработки UART

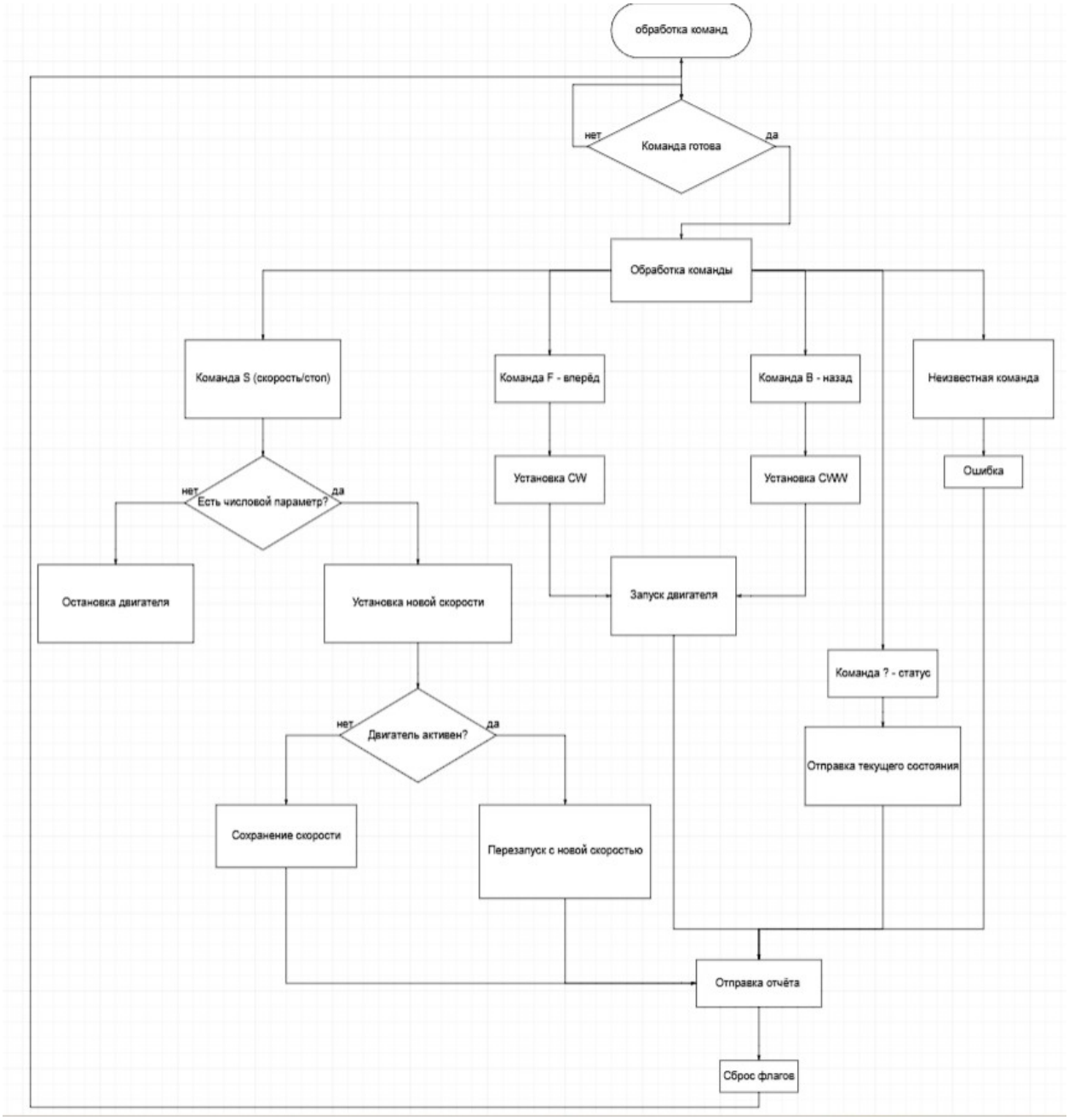

Рисунок 6 – Блок-схема обработки команд

Для работы программы были объявлены следующие переменные:

#define DIR_PIN LATBbits.LATB3 – переменная, в которой задаётся направление движения шагового двигателя, которое выводится через порт RB3.

#define CLOCK_PIN LATBbits.LATB4 – переменная, регулирующая состояние пина драйвера L297 под названием CLOCK. Через порт RB4 будет подаваться переменный сигнал, с помощью которого будет контролироваться скорость вращения двигателя.

typedef enum {

MOTOR_STOP – показывает, что двигатель остановлен.

MOTOR_CW – показывает, что двигатель движется по часовой стрелке.

MOTOR_CCW – показывает, что двигатель движется против часовой стрелке.

} MotorState;

typedef используется для того, чтобы переменная, которой MotorState присвоен этот тип, принимал только те значения, которые перечислены в скобках.

volatile MotorState motor_state = MOTOR_STOP – показывает состояние, в котором находится двигатель. Изначально двигатель не совершает движения.

volatile uint16_t current_speed = 10 – определяет скорость шагового двигателя, а именно количество шагов в секунду.

volatile uint16_t step_interval = 0 – показывает интервал между шагами в тактах таймера.

volatile uint8_t uart_buffer[32] – переменная, куда записываются команды с терминала.

volatile uint8_t uart_index = 0 – определяет положение байта в буфере

volatile uint8_t command_ready = 0 – флаг готовности команды для обработки.

Далее реализуются функции, с помощью которых работает программа.

void System_Init(). В этой функции устанавливается частота генератора 8 МГц. Также назначаются порты для соединения МК с терминалом (RC6, RC7) и порты для подключения к драйверу (RB3, RB4), которые устанавливаются цифровыми. Далее настраивается UART на скорость 9600 бод и порты UART. Также устанавливаются параметры таймера.

void SetSpeed(). В этой функции реализуется установка скорости вращения шагового двигателя. Скорость устанавливается от 1 до 200 шагов в секунду для правильной работы шагового двигателя. Скорость высчитывается

следующим образом. Один такт в МК совершается за $T_{\text{такт.}}$=4 мкс. При этом количество тактов между шагами обратно пропорционально скорости вращения двигателя (обозначим как x). Пусть скорость равна 1 шаг в секунду. Тогда время между шагами ($T_{\text{инт.}}$) равно:

$$T_{\text{инт.}} = \frac{1/x}{2T_{\text{такт.}}} = \frac{1}{2 \cdot 4\text{мкс}} = 125000 \text{ тактов}$$

Формула интервала в общем виде:

$$T_{\text{инт.}} = \frac{125000}{x} \cdot T_{\text{такт.}}$$

При изменении скорости таймер останавливается, устанавливается начальное значение таймера, равное 65535-$T_{\text{инт.}}$, затем запускается снова и работает по новой.

UART_SendChar()

Отправляет одиночный символ через последовательный интерфейс UART. Функция использует ожидание освобождения регистра передачи (флаг TRMT) перед записью нового символа, что гарантирует корректную передачу данных без потерь.

UART_SendString()

Осуществляет отправку строки символов, последовательно передавая каждый символ через вызов UART_SendChar(). Функция работает до достижения нулевого терминатора строки, что обеспечивает гибкость работы со строками различной длины.

UART_SendNumber()

Выполняет преобразование целочисленного значения в символьное представление и передачу его по UART. Алгоритм использует временный буфер для накопления цифр в обратном порядке с последующим выводом в правильной последовательности, включая обработку нулевого значения.

StartMotor()

Инициирует вращение шагового двигателя с предварительной настройкой аппаратного таймера. Функция проверяет текущее состояние

двигателя, останавливает таймер, устанавливает начальный уровень сигнала управления, конфигурирует регистры таймера для генерации импульсов с заданным интервалом и запускает счет. Применяется для начала движения и перезапуска при изменении параметров.

StopMotor()

Выполняет полную остановку двигателя путем деактивации таймера, установки состояния "остановлен" и сброса управляющего сигнала в нулевое положение. Обеспечивает безопасное прекращение работы системы.

ProcessCommand()

Обрабатывает команды управления, полученные через UART интерфейс. Функция реализует конечный автомат с поддержкой следующих операций:

- Установка скорости вращения (команда 'S' с числовым параметром)
- Остановка двигателя (команда 'S' без параметра)
- Вращение по часовой стрелке (команда 'F')
- Вращение против часовой стрелки (команда 'B')
- Запрос текущего состояния системы (команда '?')

Алгоритм включает синтаксический разбор строки команды, валидацию параметров, выполнение соответствующих действий и формирование ответных сообщений о статусе выполнения.

ISR(). Функция обработки прерывания от таймера Timer1, реализующая генерацию меандра 50% для управления шаговым двигателем. При каждом срабатывании прерывания выполняется:

- Сброс флага прерывания
- Проверка состояния двигателя (активен/остановлен)
- Инверсия состояния выходного сигнала CLOCK при активном двигателе
- Перезагрузка таймера для следующего периода

Данная реализация обеспечивает точную синхронизацию и равномерность импульсов управления независимо от нагрузки на процессор в основном цикле программы.

7 ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММЫ

Была собрана схема в Proteus 8, которая показана на рисунке 7.

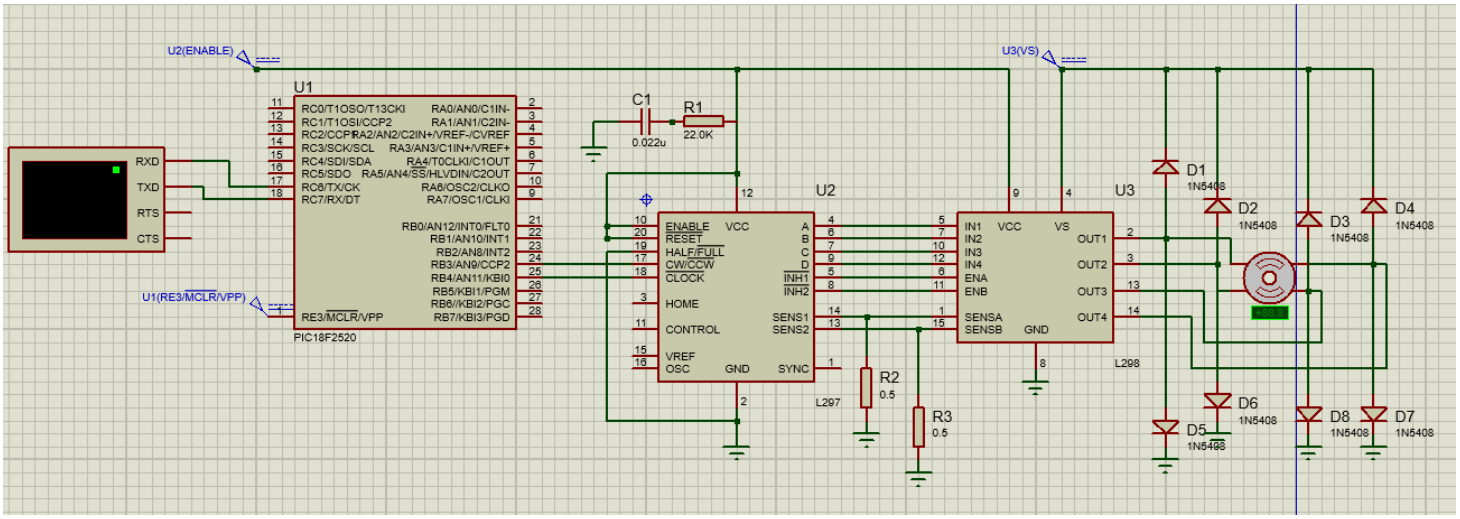

Рисунок 7 – Схема работы шагового двигателя в Proteus 8

При запуске программы терминал показывает команды, которые можно ввести. Результат показан на рисунке 8.

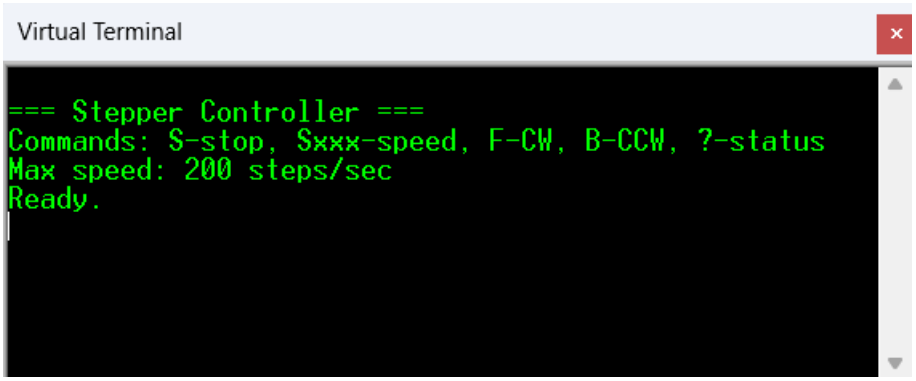

Рисунок 8 – Терминал в начале работы шагового двигателя

Результат при работе схемы показан на рисунке 9.

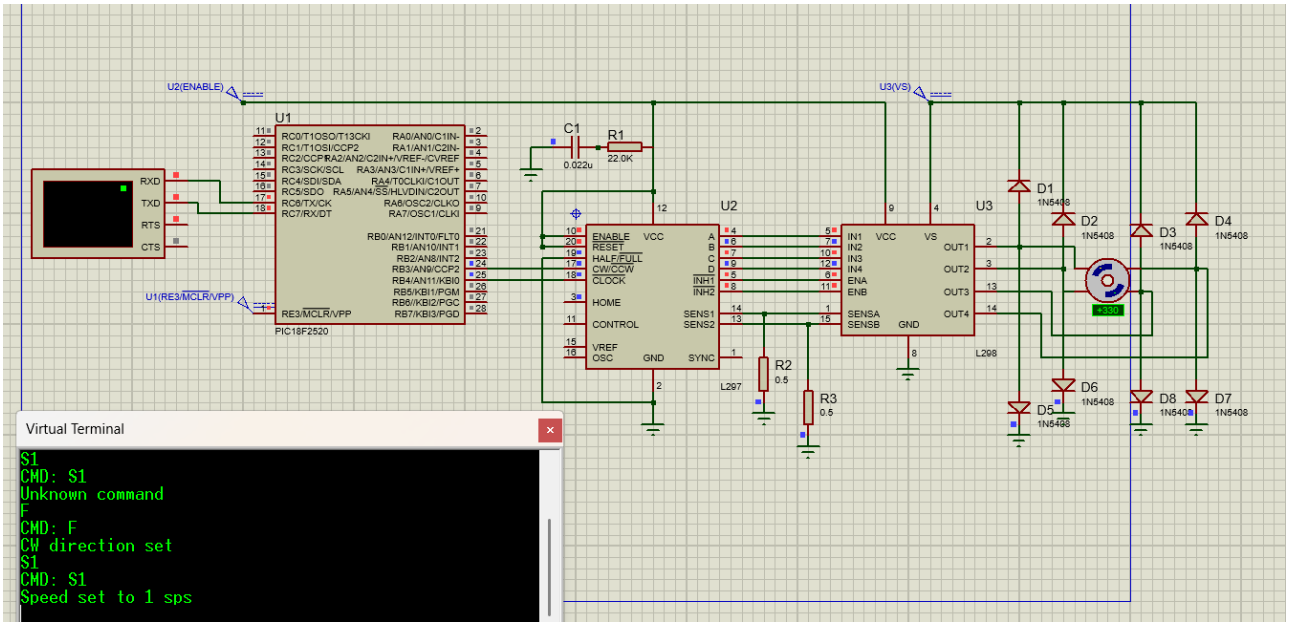

Рисунок 9 – Пример работы схемы в Proteus 8

В Proteus 8 можно менять направление вращения двигателя, но при больших скоростях ротор двигателя начинает колебаться на месте.

8 РЕАЛИЗАЦИЯ СХЕМЫ НА ПРАКТИКЕ

Схема управления шаговым двигателем была реализована физически. Схема подключения элементов друг к другу показана на рисунке 10.

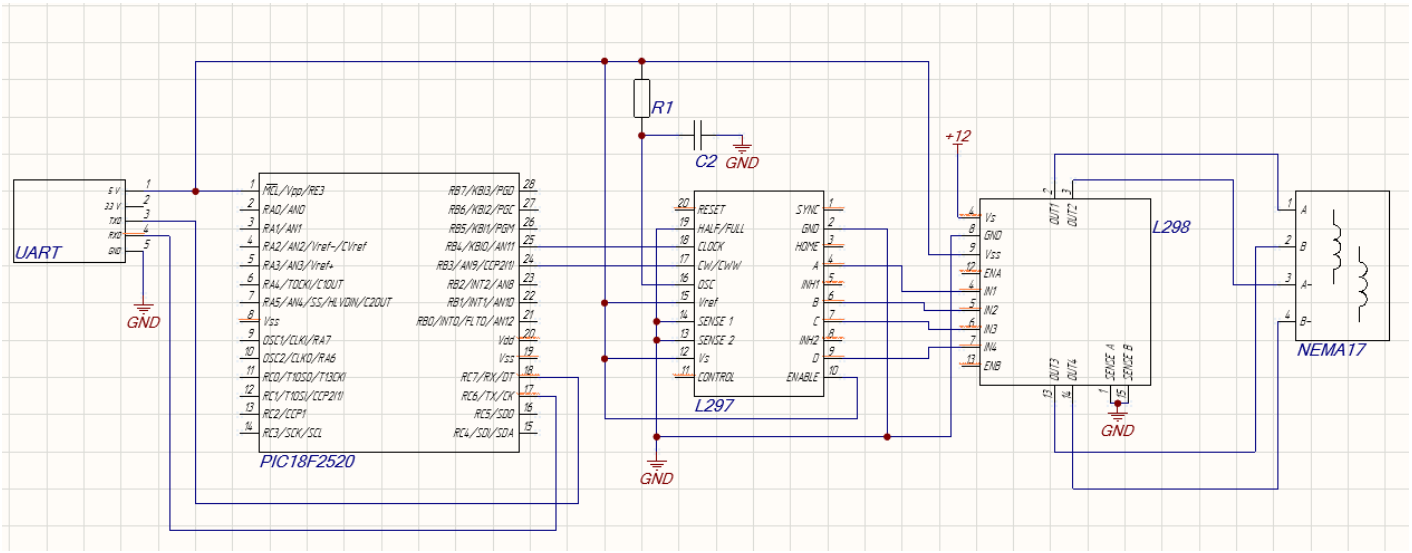

Рисунок 10 – Схема соединений элементов между собой

Собранная схема показана на рисунке 11.

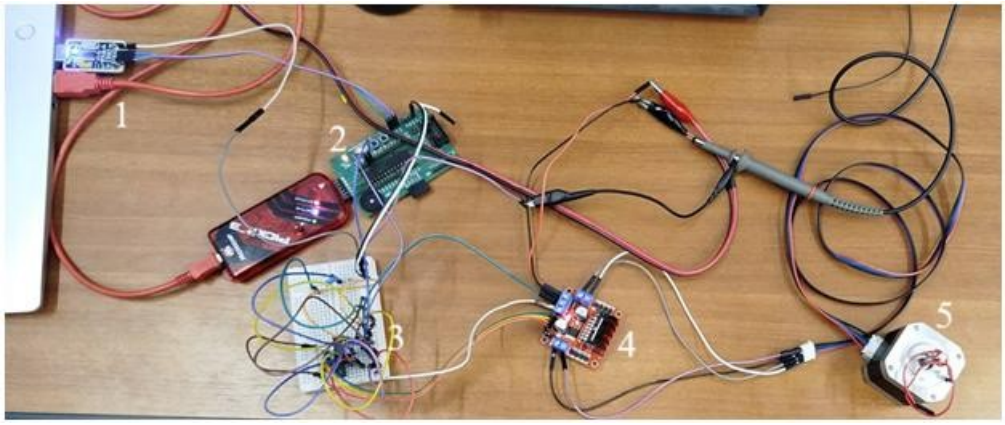

Рисунок 11 – Физическая реализация системы управления шаговым двигателем

Элементы схемы соответствуют следующим номерам:

1 – UART-адаптер

2 – Микроконтроллер PIC18F2520

3 – Драйвер L297

4 – Драйвер L298N

5 – Шаговый двигатель NEMA 17

Через ПК по UART-адаптеру на МК отправляются команды. На экране наблюдается, в каком режиме работает двигатель.

ВЫВОД

В ходе выполнения курсовой работы была разработана система управления шаговым двигателем на основе PIC18F2520. Благодаря ей есть возможность менять направление и скорость шагового двигателя с помощью ПК. Также была проведена симуляция работы шагового двигателя в среде Proteus 8. В ней удалось менять направление движения в реальном времени, но с изменением скорости возникли проблемы, т. к. на высоких скоростях ротор двигателя колебался на месте. Также схема была реализована физически.

ПРИЛОЖЕНИЕ

Листинг программы

```
#pragma config OSC = INTIO67      // Oscillator Selection bits (Internal oscillator block, port function on RA6 and RA7)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)


#pragma  config PWRT = OFF        // Power-up Timer Enable  bit (PWRT disabled)
#pragma config BOREN = SBORDIS   // Brown-out Reset Enable bits (Brown-out Reset enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 3         // Brown Out Reset Voltage bits (Minimum setting)

#pragma config WDT = OFF        // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)

#pragma config CCP2MX = PORTC   // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBADEN = OFF     // PORTB A/D Enable bit (PORTB<4:0> pins are configured as digital I/O on Reset)
#pragma config LPT1OSC = OFF    // Low-Power Timer1 Oscillator Enable bit (Timer1 configured for higher power operation)
#pragma  config MCLRE  =  ON        // MCLR Pin  Enable bit (MCLR pin enabled; RE3 input pin disabled)
```

#pragma config STVREN = ON   // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = OFF   // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config XINST = OFF

#pragma config OSC = INTIO67   // Oscillator Selection bits (Internal oscillator block, port function on RA6 and RA7)
#pragma config FCMEN = OFF   // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF   // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)

#pragma config PWRT = OFF   // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = SBORDIS   // Brown-out Reset Enable bits (Brown-out Reset enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 3   // Brown Out Reset Voltage bits (Minimum setting)

#pragma config WDT = OFF   // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))
#pragma config WDTPS = 32768   // Watchdog Timer Postscale Select bits (1:32768)

#pragma config CCP2MX = PORTC   // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)

```
#pragma config PBADEN = OFF     // PORTB A/D Enable bit (PORTB<4:0> pins are configured as digital I/O on Reset)
#pragma config LPT1OSC = OFF    // Low-Power Timer1 Oscillator Enable bit (Timer1 configured for higher power operation)
#pragma  config MCLRE  =  ON       // MCLR Pin  Enable bit (MCLR pin enabled; RE3 input pin disabled)

#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = OFF        // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config XINST = OFF



#include <xc.h>
#include <stdint.h>

// Пины для управления L297
#define DIR_PIN      LATBbits.LATB3    // Направление (RB3)
#define CLOCK_PIN    LATBbits.LATB4    // Шаговые импульсы (RB4)

typedef enum {
   MOTOR_STOP,
   MOTOR_CW,      // По часовой стрелке
   MOTOR_CCW      // Против часовой стрелки
} MotorState;

// Глобальные переменные
volatile MotorState motor_state = MOTOR_STOP;
volatile uint16_t current_speed = 10;   // Текущая скорость в шагах/сек
```

```
volatile uint16_t step_interval = 0;    // Интервал между шагами в тиках
volatile uint8_t uart_buffer[32];
volatile uint8_t uart_index = 0;
volatile uint8_t command_ready = 0;

// Прототипы функций
void UART_SendChar(char data);
void UART_SendString(const char *str);
void UART_SendNumber(uint16_t num);

void System_Init(void) {
    // Тактирование - 8 МГц
    OSCCON = 0x70;

    // Отключение аналоговых функций
    ADCON0 = 0x00;
    ADCON1 = 0x0F;
    CMCON = 0x07;

    // UART пины
    TRISCbits.TRISC6 = 0;
    TRISCbits.TRISC7 = 1;

    // Управляющие пины
    TRISBbits.TRISB3 = 0;
    TRISBbits.TRISB4 = 0;

    // Начальное состояние
    DIR_PIN = 0;
    CLOCK_PIN = 0;
```

```
    // UART 9600 бод
    TXSTAbits.SYNC = 0;
    TXSTAbits.BRGH = 1;
    BAUDCONbits.BRG16 = 0;
    SPBRG = 51;

    RCSTAbits.SPEN = 1;
    RCSTAbits.CREN = 1;
    TXSTAbits.TXEN = 1;

    // Настройка Timer1 с предделителем 1:8
    T1CON = 0x31;          // 1:8 предделитель
    TMR1H = 0x00;
    TMR1L = 0x00;
    PIR1bits.TMR1IF = 0;
    T1CONbits.TMR1ON = 0;

    // Включаем прерывания
    INTCONbits.PEIE = 1;
    INTCONbits.GIE = 1;
    PIE1bits.TMR1IE = 1;
}

// Функция установки скорости
void SetSpeed(uint16_t steps_per_second) {
    uint32_t temp;

    if (steps_per_second < 1) steps_per_second = 1;
    if (steps_per_second > 200) steps_per_second = 200;
```

```
    // Расчет интервала между шагами для Timer1 с предделителем 1:8
    temp = 125000UL / steps_per_second;

    // Ограничения
    if (temp > 65535UL) temp = 65535UL;
    if (temp < 100UL) temp = 100UL;

    step_interval = (uint16_t)temp;
    current_speed = steps_per_second;

    // Если таймер работает, перезагружаем его
    if (T1CONbits.TMR1ON) {
        T1CONbits.TMR1ON = 0;
        TMR1H = (uint8_t)((65535 - step_interval) >> 8);
        TMR1L = (uint8_t)(65535 - step_interval);
        T1CONbits.TMR1ON = 1;
    }
}

void UART_SendChar(char data) {
    while(!TXSTAbits.TRMT);
    TXREG = data;
}

void UART_SendString(const char *str) {
    while(*str) {
        UART_SendChar(*str++);
    }
}
```

```
void UART_SendNumber(uint16_t num) {
    char buffer[6];
    uint8_t i = 0;

    if (num == 0) {
        UART_SendChar('0');
        return;
    }

    while (num > 0) {
        buffer[i++] = '0' + (num % 10);
        num /= 10;
    }

    while (i > 0) {
        UART_SendChar(buffer[--i]);
    }
}

void StartMotor(void) {
    if (motor_state != MOTOR_STOP) {
        // Останавливаем таймер
        T1CONbits.TMR1ON = 0;

        // Начинаем с низкого уровня на CLOCK
        CLOCK_PIN = 0;

        // Устанавливаем начальное значение таймера
        TMR1H = (uint8_t)((65535 - step_interval) >> 8);
```

```
        TMR1L = (uint8_t)(65535 - step_interval);

        // Сбрасываем флаг прерывания
        PIR1bits.TMR1IF = 0;

        // Запускаем таймер
        T1CONbits.TMR1ON = 1;
    }
}

void StopMotor(void) {
    // Останавливаем таймер
    T1CONbits.TMR1ON = 0;
    motor_state = MOTOR_STOP;
    CLOCK_PIN = 0;
}

void ProcessCommand(void) {
    if (!command_ready) return;

    UART_SendString("CMD: ");
    UART_SendString((char*)uart_buffer);
    UART_SendString("\r\n");

    switch(uart_buffer[0]) {
        case 'S':
        case 's':
            if (uart_buffer[1] >= '0' && uart_buffer[1] <= '9') {
                uint16_t speed = 0;
                uint8_t i = 1;
```

```
            while (uart_buffer[i] >= '0' && uart_buffer[i] <= '9') {
                speed = speed * 10 + (uart_buffer[i] - '0');
                i++;
            }
            SetSpeed(speed);
            UART_SendString("Speed set to ");
            UART_SendNumber(speed);
            UART_SendString(" sps\r\n");

            // Если двигатель уже работает, перезапускаем с новой скоростью
            if (motor_state != MOTOR_STOP) {
                StartMotor();
            }
        } else {
            // Полная остановка
            StopMotor();
            UART_SendString("STOPPED\r\n");
        }
        break;

    case 'F':
    case 'f':
        if (motor_state != MOTOR_CW) {
            motor_state = MOTOR_CW;
            DIR_PIN = 1;  // CW
            UART_SendString("CW direction set\r\n");
            StartMotor();
        }
        break;
```

```
        case 'B':
        case 'b':
            if (motor_state != MOTOR_CCW) {
                motor_state = MOTOR_CCW;
                DIR_PIN = 0;  // CCW
                UART_SendString("CCW direction set\r\n");
                StartMotor();
            }
            break;

        case '?':
            UART_SendString("Status: ");
            if (motor_state == MOTOR_STOP) UART_SendString("STOPPED");
            else if (motor_state == MOTOR_CW) UART_SendString("CW");
            else UART_SendString("CCW");
            UART_SendString("\r\nSpeed: ");
            UART_SendNumber(current_speed);
            UART_SendString(" sps\r\n");
            break;

        default:
            UART_SendString("Unknown command\r\n");
            break;
    }

    command_ready = 0;
    uart_index = 0;
}
```

```
// Прерывание Timer1 - простой меандр 50%
void __interrupt() ISR(void) {
    static uint8_t clock_state = 0;

    if (PIR1bits.TMR1IF) {
        PIR1bits.TMR1IF = 0;

        if (motor_state != MOTOR_STOP) {
            // Просто переключаем CLOCK (меандр 50%)
            clock_state = !clock_state;
            CLOCK_PIN = clock_state;

            // Перезагружаем таймер
            TMR1H = (uint8_t)((65535 - step_interval) >> 8);
            TMR1L = (uint8_t)(65535 - step_interval);
        }
    }
}

void main(void) {
    System_Init();

    // Установка начальной скорости 10 шагов/сек
    SetSpeed(10);

    UART_SendString("\r\n=== Stepper Controller ===\r\n");
    UART_SendString("Commands: S-stop, Sxxx-speed, F-CW, B-CCW, ?-
status\r\n");
    UART_SendString("Max speed: 200 steps/sec\r\n");
```

```
    UART_SendString("Ready.\r\n");

    while(1) {
        // Обработка UART
        if (PIR1bits.RCIF) {
            uint8_t data = RCREG;

            if (RCSTAbits.OERR) {
                RCSTAbits.CREN = 0;
                RCSTAbits.CREN = 1;
                continue;
            }

            if (data == '\r' || data == '\n') {
                if (uart_index > 0) {
                    uart_buffer[uart_index] = '\0';
                    command_ready = 1;
                }
            } else if (uart_index < 31) {
                uart_buffer[uart_index++] = data;
            }

            PIR1bits.RCIF = 0;
        }

        if (command_ready) {
            ProcessCommand();
        }
    }
}
```