Лабораторная работа № 17.

Тема: Разработка системы управления двигателем.

Цель: Ознакомиться с основами управления двигателями постоянного

Теоретическая часть:

Типы двигателей постоянного тока

Двигатели постоянного тока (DC) являются основными активными элементами во множестве проектов на Arduino, начиная от робототехники до автоматизированных устройств. Вот основные типы DC моторов, которые часто используются в различных проектах:

1. **Простые DC моторы**: Это базовые моторы, которые вращаются при подаче напряжения. Скорость вращения мотора зависит от напряжения, а направление вращения — от полярности подключения.
2. **Моторы с редуктором (Gear Motors)**: Эти моторы имеют встроенный редуктор, который снижает скорость вращения вала, но при этом увеличивает крутящий момент. Они подходят для задач, где необходима высокая сила при низкой скорости.
3. **Серводвигатели**: Хотя технически это не классические DC моторы, серводвигатели часто используются там, где необходимо точное позиционирование. Они получают сигналы управления, которые задают угол поворота вала.
4. **Шаговые двигатели**: Используются для точного контроля положения вала, вращаясь на фиксированный угол (шаг) с каждым электрическим импульсом. Не являются DC моторами в классическом смысле, но часто встречаются в похожих приложениях.
5. **Бесколлекторные DC моторы (BLDC)**: Эти моторы обеспечивают более высокую эффективность и продолжительность работы по сравнению с обычными коллекторными моторами. Они требуют специальных схем управления для работы, так как используют электронное коммутирование.
6. **Вибромоторы**: Используются в приложениях, где требуется генерация вибрации, например, в мобильных устройствах или игрушках.

Принципы управления скоростью и направлением вращения

Каждый тип мотора имеет свои особенности и области применения. Выбор подходящего типа мотора зависит от требований проекта,

включая необходимую скорость, крутящий момент, управление и другие характеристики. В проектах на Arduino часто используются дополнительные компоненты для управления этими моторами, такие как драйверы моторов или шилды, что позволяет легко интегрировать моторы в электронные схемы.

Для управления двигателями постоянного тока в проектах на Arduino (микроконтроллерах) обычно используются различные методы для регулирования скорости и изменения направления вращения. Вот основные принципы и методы, которые применяются для этого:

Управление скоростью

1. **Широтно-импульсная модуляция (PWM)**: PWM – это метод, который позволяет контролировать среднее напряжение, подаваемое на двигатель, изменяя длительность импульсов электрического тока. Arduino предоставляет встроенные функции для генерации PWM-сигналов, которые можно использовать для управления скоростью двигателя.
2. **Аналоговое регулирование**: В некоторых случаях можно использовать переменные резисторы или потенциометры для аналогового контроля напряжения питания мотора, но это менее эффективно и редко используется по сравнению с PWM.

Управление направлением

1. **Использование реле**: Можно использовать пару реле для изменения полярности напряжения на моторе, что позволит изменять направление вращения. При этом одно реле контролирует сам мотор, а второе — направление вращения.
2. **Использование H-моста**: H-мост — это схема, которая позволяет легко реверсировать напряжение, подаваемое на двигатель, и таким образом изменять направление вращения. Это самый популярный метод управления направлением для DC моторов в проектах на Arduino. Микросхемы H-моста, такие как L298N или L293D, широко доступны и легко интегрируются с Arduino.

Интеграция с Arduino

Для интеграции контроля мотора с Arduino обычно требуется следующее:

- **Драйвер мотора**: Это может быть драйвер на базе H-моста или простой транзистор/мосфет для однонаправленного управления.

- **Программное обеспечение**: Написание кода для Arduino, который будет генерировать PWM сигналы и контролировать логические уровни на входах драйвера мотора для управления скоростью и направлением.
- **Защита**: Важно также учитывать защиту от перегрузок и помех, что может включать в себя добавление диодов, предохранителей и конденсаторов.

Шаговые двигатели

Шаговые двигатели — это тип электродвигателя, который перемещается по шагам, делая фиксированный угол поворота с каждым электрическим импульсом. Они особенно полезны в приложениях, где требуется точное управление положением, например, в 3D-принтерах, ЧПУ машинах и роботизированных системах.

Принципы работы шаговых двигателей

Шаговые двигатели состоят из статора и ротора. Статор обычно содержит несколько катушек, которые при подаче на них напряжения создают магнитные поля, вызывая вращение ротора на определённый угол. Ротор изготовлен из магнитомягкого материала и может содержать постоянные магниты или быть без магнитов.

1. **Различие по конструкции:**
    - **Униполярные двигатели** имеют катушки, каждая из которых может быть включена индивидуально.
    - **Биполярные двигатели** требуют изменения направления тока через катушки для изменения полярности магнитного поля.
2. **Типы шаговых двигателей:**
    - **Двигатели с постоянными магнитами:** Ротор содержит постоянные магниты, и вращение достигается за счёт магнитного взаимодействия с электромагнитными катушками статора.
    - **Гибридные двигатели:** Комбинируют характеристики двигателей с переменным и постоянным магнитом, предлагая высокую производительность и точность.

Драйверы шаговых двигателей

Для управления шаговыми двигателями используются специальные драйверы, которые обеспечивают необходимые последовательности тока для перемещения мотора на желаемое количество шагов. Некоторые популярные драйверы:

- **ULN2003**: Простой драйвер для униполярных шаговых двигателей.
- **A4988**: Многофункциональный драйвер, позволяющий управлять как униполярными, так и биполярными шаговыми двигателями. Он поддерживает микрошаговое управление, что позволяет увеличить точность и плавность движения.
- **DRV8825**: Похож на A4988, но способен обеспечивать больший ток и поддерживает большее разрешение микрошагов.
- **L297** — это устройство для управления шаговыми двигателями, разработанное для использования совместно с внешними переключателями мощности, такими как транзисторы или мощные драйверы, например, L298. L297 предназначен для управления как биполярными, так и униполярными шаговыми двигателями и обеспечивает широкий набор функций для управления двигателем, включая выбор режима шага, управление фазами и поддержку микрошагования.

Программирование и управление

Шаговые двигатели управляются путём отправки серии электрических импульсов (шагов), которые активируют катушки двигателя в точной последовательности. Arduino может генерировать эти сигналы через свои выходные пины, управляя драйвером шагового двигателя. В Arduino есть библиотеки, такие как **Stepper**, которые упрощают создание программ для управления шаговыми двигателями.

Использование драйвера не только упрощает управление мотором, но и снижает нагрузку на микроконтроллер, так как основная часть работы по управлению мощностью и последовательностями катушек выполняется драйвером.

Библиотеки для управления двигателями.

В Atmel Studio вы можете использовать библиотеки для управления двигателями, которые предоставляют функции для работы с различными типами двигателей, такими как шаговые, постоянного тока и серводвигатели. Ниже приведены некоторые популярные библиотеки для управления двигателями в Atmel Studio:

AVR Motor Control Library: Это библиотека, разработанная компанией Atmel, предоставляющая возможности управления

различными типами двигателей на микроконтроллерах AVR. Она включает в себя функции для управления скоростью и направлением вращения двигателей, а также функции обратной связи.

AVR-libc: Это стандартная библиотека для микроконтроллеров AVR, которая включает в себя набор функций для работы с различными периферийными устройствами, включая двигатели. Она обеспечивает доступ к низкоуровневым функциям управления портами ввода-вывода, таймерами и прерываниями, что позволяет создавать пользовательские решения для управления двигателями.

Servo.h: Это стандартная библиотека Arduino IDE для управления серводвигателями. Вы можете использовать ее в Atmel Studio, если пишете код на языке C или C++ для микроконтроллеров AVR. Она предоставляет функции для установки положения серводвигателя и настройки его параметров.

Stepper.h: Это также стандартная библиотека Arduino IDE для управления шаговыми двигателями. Она предоставляет функции для управления скоростью и направлением вращения шагового двигателя, а также для выполнения шаговых движений с заданным количеством шагов.

Драйвер L297: подключение и управление шаговым двигателем

L297 - это популярный микросхемный драйвер шаговых двигателей, который используется для управления биполярными шаговыми двигателями.

Подключение:

1. **Питание:**
   - VCC подключается к источнику питания +5 В.
   - GND подключается к земле.
2. **Управление:**
   - STEP: к этому пину подключается сигнал управления шагами.
   - DIR: к этому пину подключается сигнал направления вращения.
   - RS: к этому пину (если используется) подключается сигнал выбора режима (половинный/полный шаг).
3. **Двигатель:**
   - A0, A1, A2, A3: к этим пинам подключаются катушки шагового двигателя.

Принцип управления:

Драйвер L297 управляет шаговым двигателем, переключая ток в катушках двигателя в соответствии с последовательностью шагов.

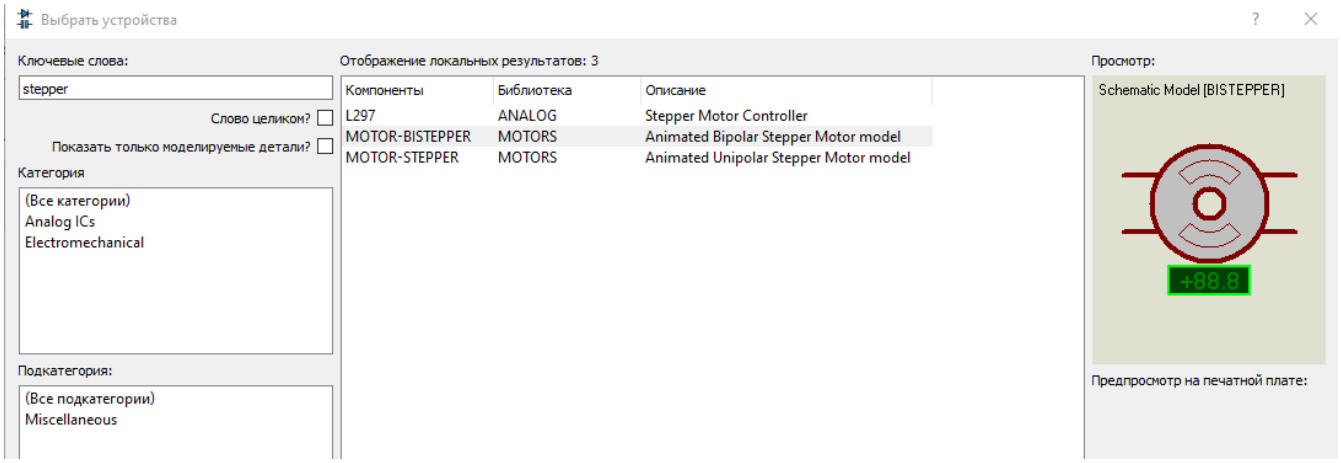
- **Последовательность шагов:** Определяет порядок подачи тока на катушки двигателя.
- **Режим:**
  - **Полный шаг:** Двигатель делает один шаг за один цикл подачи тока.
  - **Половинка шага:** Двигатель делает два шага за один цикл подачи тока.

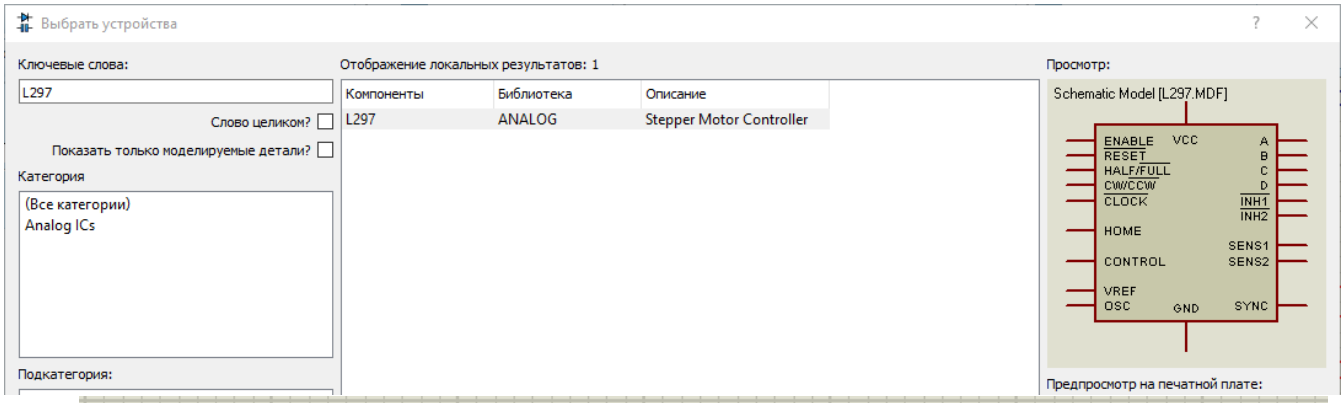
Сигналы управления:

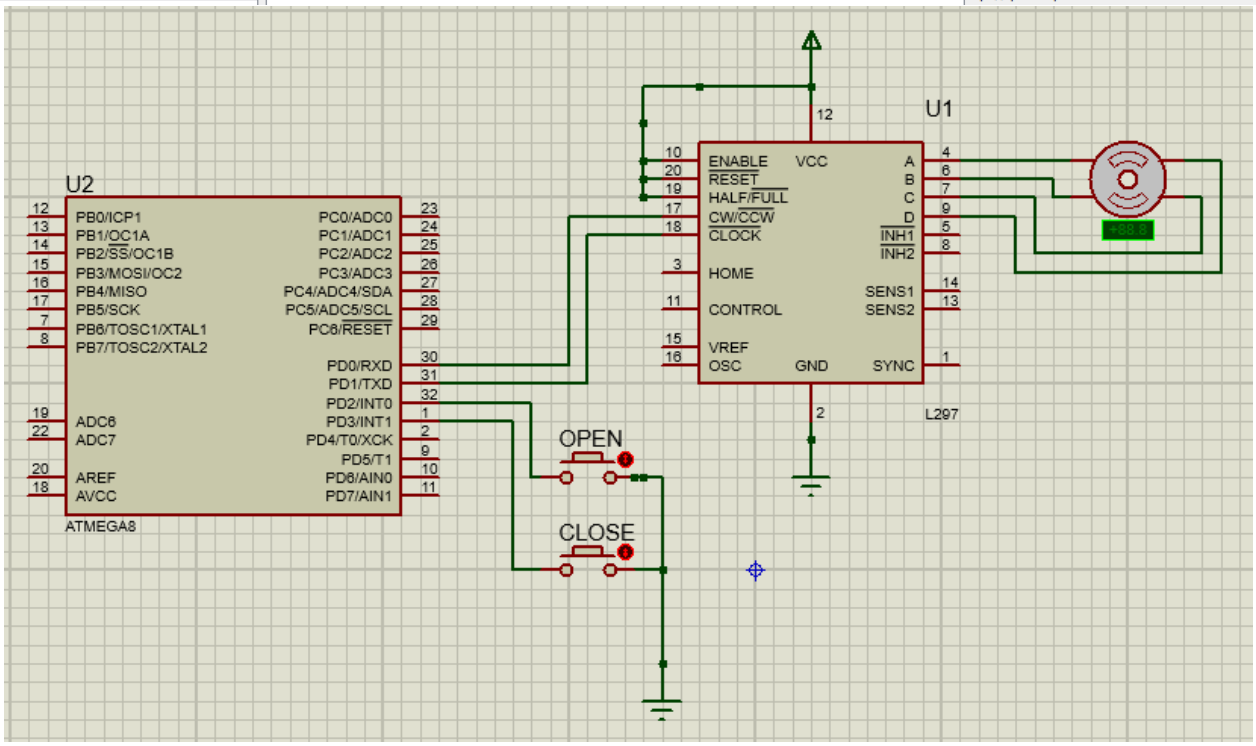
- **STEP:** Каждый раз, когда на этот пин подается импульс, двигатель делает один шаг.
- **DIR:**
  - Логический 0: вращение по часовой стрелке.
  - Логический 1: вращение против часовой стрелки.
- **RS (опционально):**
  - Логический 0: полный шаг.
  - Логический 1: половинный шаг.

Пример использование шагового двигателя в Arduino.

Соберем схему в Proteus:







Ниже представлен для управления шаговым который осуществляет открытие и закрытие штор по командам с кнопок.

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>

#define DIR_PORT PORTD // опр. макроса DIR_PORT и заменяет его на PORTD. понятное имя DIR_PORT вместо PORTD
#define DIR_PIN 0 // 0-й бит порта C управляет направлением вращения двигателя.
#define STEP_PORT PORTD
#define STEP_PIN 1 // 1-й бит порта C управляет отдельным шагом двигателя.
#define OPEN_BUTTON_PIN 2 // "Открыть" шторы.
#define CLOSE_BUTTON_PIN 3  // "Закрыть" шторы.
```

```
#define DIR_MASK (1 << DIR_PIN)
#define STEP_MASK (1 << STEP_PIN)
#define OPEN_BUTTON_MASK (1 << OPEN_BUTTON_PIN)
#define CLOSE_BUTTON_MASK (1 << CLOSE_BUTTON_PIN)

#define OPEN_STEPS 10 // Количество шагов для открытия
#define CLOSE_STEPS 10 // Количество шагов для закрытия

#define OPENED_FLAG 1 // Флаг открытого состояния
#define CLOSED_FLAG 0 // Флаг закрытого состояния

int current_state = CLOSED_FLAG; // Текущее состояние
(открыто/закрыто)

void initPorts() {
	// Настройка портов управления L297
	DDRD |= DIR_MASK | STEP_MASK;
	DIR_PORT &= ~DIR_MASK;
	STEP_PORT &= ~STEP_MASK;

	// Настройка портов входов для кнопок
	DDRD &= ~(OPEN_BUTTON_MASK | CLOSE_BUTTON_MASK);
	PORTD |= OPEN_BUTTON_MASK | CLOSE_BUTTON_MASK;
}

void stepMotor(int steps, int direction) {
	if (direction == 1) {
		DIR_PORT |= DIR_MASK;
		} else {
		DIR_PORT &= ~DIR_MASK;
	}

	int steps_taken = 0;
	while (steps_taken != abs(steps)) {
		STEP_PORT |= STEP_MASK;
		_delay_ms(50);
		STEP_PORT &= ~STEP_MASK;
		_delay_ms(50);

		steps_taken++;
	}
}

void handleManualControls() {
```

```
        // Проверка нажатия кнопки "Открыть шторы"
        if (!(PIND & OPEN_BUTTON_MASK)) {
                if (current_state == CLOSED_FLAG) {
                        stepMotor(OPEN_STEPS, 1);
                        current_state = OPENED_FLAG;
                }
        }

        // Проверка нажатия кнопки "Закрыть шторы"
        if (!(PIND & CLOSE_BUTTON_MASK)) {
                if (current_state == OPENED_FLAG) {
                        stepMotor(OPEN_STEPS, -1);
                        current_state = CLOSED_FLAG;
                }
        }
}

int main(void) {
        initPorts();

        while (1) {
                handleManualControls();
                _delay_ms(100);
        }

        return 0;
}
```

Пояснения к коду:

Определения и настройки

- **#define F_CPU 8000000UL**: Задает частоту процессора как 8 МГц, что необходимо для функций задержки.
- **#include <avr/io.h>**, **<util/delay.h>**, **<stdlib.h>**: Подключение стандартных библиотек для работы с портами ввода-вывода, задержками и стандартными функциями языка C.
- **#define** макросы для портов и пинов:
  - **DIR_PORT**, **STEP_PORT** определяют порты для управления направлением и шагами двигателя.
  - **DIR_PIN**, **STEP_PIN** указывают конкретные пины для управления направлением и шагами.
  - **OPEN_BUTTON_PIN**, **CLOSE_BUTTON_PIN** задают пины для кнопок открытия и закрытия.
- **#define** маски:

- **DIR_MASK**, **STEP_MASK** создают битовые маски для управления направлением и шагами.
- **OPEN_BUTTON_MASK**, **CLOSE_BUTTON_MASK** создают битовые маски для чтения состояний кнопок.

Функции

- **initPorts()**: Настраивает направление портов (ввод/вывод), начальные состояния для управления двигателем и активирует подтягивающие резисторы для кнопок.
- **stepMotor(int steps, int direction)**: Выполняет заданное количество шагов двигателя в указанном направлении. Использует задержки между шагами для контроля скорости.
- **handleManualControls()**: Проверяет состояние кнопок и инициирует движение двигателя в зависимости от текущего состояния (открыто или закрыто).

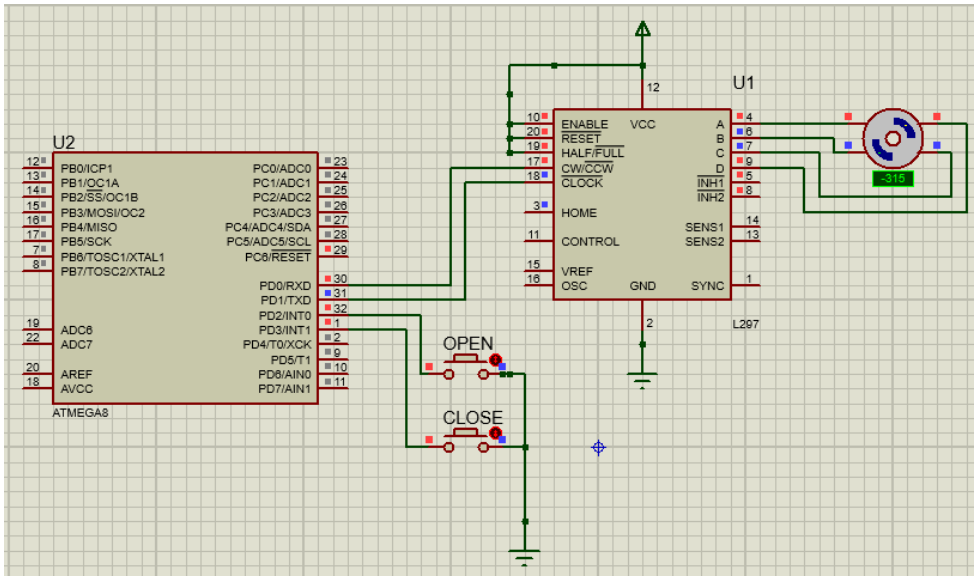
Основной цикл

- **main()**:
  - Инициализирует порты.
  - В бесконечном цикле проверяет состояние кнопок и обрабатывает управление двигателем.
  - Включает небольшую задержку для устранения дребезга контактов кнопок.

Логика работы

Кнопки для открытия и закрытия подключены так, что при нажатии логический уровень на соответствующих пинах становится низким (используется внутренний подтягивающий резистор, активированный записью HIGH в регистр PORTD). Функция **handleManualControls()** реагирует на это изменение, запуская функцию **stepMotor()** с нужным количеством шагов и направлением. Управление состоянием реализовано с помощью переменной **current_state**, что позволяет избежать повторного выполнения операции, если шторы уже находятся в нужном состоянии.

Симуляция:



Задание:

1. Соберите схему в Proteus представленную выше. Напишите и скомпилируйте код, загрузите его в микроконтроллер. Проверьте работоспособность в симуляторе.

2. В Proteus добавьте в схему датчик освещенности. Шторы должны автоматически открываться и закрываться, в зависимости от уровня освещенности. Также должна оставаться возможность управлять шторами вручную, при нажатой кнопке «Mode».