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

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 **ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

по дисциплине «Микропроцессорные системы»

НА ТЕМУ:

Контроллер беговой дорожки

Студент	ИУ6-73Б		И.А. Евсеенков
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Руководитель			И.Б. Трамов
		(Подпись, дата)	(И.О. Фамилия)

2024 г.

ЛИСТ ЗАДАНИЯ

РЕФЕРАТ

РПЗ 53 страницы, 22 рисунка, 13 таблиц, 17 источников, 3 приложения.

МИКРОКОНТРОЛЛЕР, СИСТЕМА, ДВИГАТЕЛЬ, БЕГ, ШИМ, BLUETOOTH, LCD

Объектом разработки является контроллер беговой дорожки.

Цель работы – создание функционального устройства ограниченной сложности, модель устройства и разработка необходимой документации на объект разработки.

Поставленная цель достигается посредством использования Proteus 8.

В процессе работы над курсовым проектом решаются следующие задачи: выбор МК и драйвера обмена данных, создание функциональной и принципиальной схем системы, расчет потребляемой мощности устройства, разработка алгоритма управления и соответствующей программы МК, а также написание сопутствующей документации.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК – микроконтроллер.

ТЗ – техническое задание.

Proteus 8 – пакет программ для автоматизированного проектирования (САПР) электронных схем.

LCD – жидкокристаллический дисплей.

USART – Universal Synchronous-Asynchronous Receiver/Transmitter – последовательный универсальный синхронный/асинхронный приемопередатчик.

ШИМ – широтно-импульсная модуляция, метод управления уровнем мощности, подаваемой к нагрузке, заключающийся в изменении продолжительности импульса при постоянной частоте их следования.

Bluetooth – стандарт беспроводной связи, обеспечивающий обмен данными между устройствами на основе ультракоротких радиоволн.

ВВЕДЕНИЕ

В данной работе производится разработка контроллера беговой дорожки.

В процессе выполнения работы проведён анализ технического задания, создана концепция устройства, разработаны электрические схемы, построен алгоритм и управляющая программа для МК, выполнено интерактивное моделирование устройства.

Система состоит из МК с записанной в него программой, одного обычного электродвигателя, одного шагового двигателя, драйверов для управления этими двигателями, панели управления, состоящей из кнопок, одного LCD для отображения информации о текущей тренировке, трех светоиндикаторов, которые говорят о текущем режиме тренировки, и модуле беспроводной передачи данных для передачи результатов на телефон.

Актуальность разрабатываемой модели микроконтроллера беговой дорожки заключается в том, что это довольно популярное устройство для поддержания здоровья, его восстановления и улучшения физической формы. В настоящее время на рынке спортивных тренажеров, таких как беговые дорожки, наблюдается негативная тенденция неточности выдаваемой информации о каждом занятии, так как результат зависит от множества факторов. Поскольку пользователь непосредственно взаимодействует с устройством, оно должно отвечать заявленным техническим требованиям и требованиям безопасности.

1 Конструкторская часть

1.1 Анализ требований и принцип работы системы

Исходя из требований, изложенных в техническом задании, можно сделать вывод, что устройство представляет собой беговую дорожку с возможностью регулировки скорости вращения бегового полотна, его наклона. Также должен поддерживаться учет времени тренировки, потраченных пользователем за прошедшее время калорий и отображение этих данных с последующей передачей результатов по беспроводному протоколу Bluetooth.

Модель беговой дорожки является обычным устройством для поддержания здоровой физической формы пользователя. Так как пользователь непосредственно взаимодействует с прибором, должны обязательно выполняться требования к безопасности такие, как:

- низкое время задержки;
- защитные функции;
- качественная обратная связь;
- точный расчет параметров пользователя и подаваемой нагрузки.

Принцип работы разрабатываемого устройства можно описать следующим образом.

Для начала работы устройства необходимо подать питание от сети. Информация о начале работы и приветственное сообщение отображается на ЖК-дисплее. Запуск и начало тренировки осуществляется при нажатии на кнопку «старт». На экране дисплея начинают отображаться важные данные: скорость, наклон, время тренировки и потраченные калории.

В процессе тренировки у пользователя есть возможность взаимодействовать с панелью управления и регулировать текущую скорость (кнопки «скорость +» и «скорость -»), наклон (кнопки «наклон +» и «наклон -») или выбрать одну из трех доступных программ. Доступные режимы представлены в таблице 1.

Таблица 1 – Режимы работы беговой дорожки

Номер режима	Название режима	Описание
0	Ручной	Скорость и наклон равны нулю, нагрузка регулируется пользователем
1	Ходьба	Скорость равна 4 ступени, наклон равен 8 ступени
2	Бег	Скорость равна 8 ступени, наклон равен 4 ступени

Завершить тренировку можно нажатием на кнопку «стоп», после чего беговое полотно остановится, а наклон уменьшится до нуля – беговая дорожка вернется в исходное положение.

После завершения тренировки данные о ней передаются по беспроводному протоколу передачи данных через Bluetooth-модуль на телефон.

Разработанная структурная схема модели перекрестка представлена на рисунке 1.

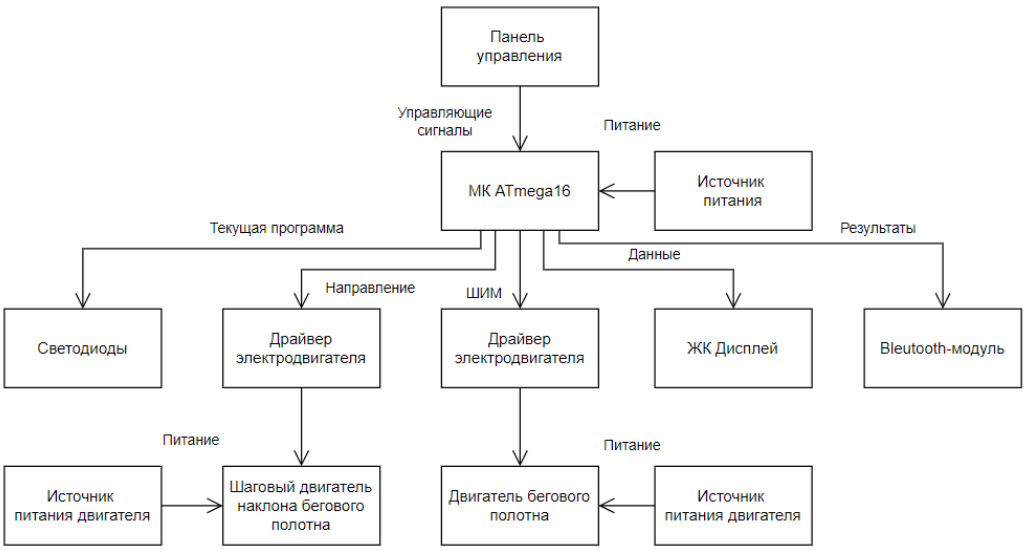

Рисунок 1 – Структурная схема устройства

1.2 Проектирование функциональной схемы

В этом разделе приведено функциональное описание работы системы и проектирование функциональной схемы.

1.2.1 Микроконтроллер ATmega16

Основным элементом разрабатываемого устройства является микроконтроллер. Существует множество семейств МК, для разработки выберем семейство AVR, указанное в ТЗ.

Семейство AVR – это группа 8-битных микроконтроллеров, разработанных компанией Atmel (в настоящее время часть Microchip Technology). Эти микроконтроллеры получили широкую популярность благодаря своей простоте, надежности и высокой производительности. AVR-микроконтроллеры используются в различных приложениях, от простых устройств до сложных систем управления [1].

AVR использует архитектуру RISC (Reduced Instruction Set Computing), что позволяет выполнять инструкции за один такт. Это обеспечивает высокую производительность при относительно низком потреблении энергии.

В микроконтроллерах AVR встречаются следующие типы памяти:

- Flash-память, используемая для хранения программного кода;
- SRAM (Static Random Access Memory) – полупроводниковая оперативная память, в которой каждый двоичный или троичный разряд хранится в схеме с положительной обратной связью; предназначена для временного хранения данных во время выполнения программы;
- EEPROM (Electrically Erasable Programmable Read-Only-Memory) – энергонезависимая память для хранения данных, которые должны сохраняться даже при отключении питания.

AVR-микроконтроллеры имеют множество портов ввода/вывода, которые можно настраивать для различных функций, таких как цифровые входы/выходы, аналоговые входы и специальные функции (например, ШИМ).

Микроконтроллеры поддерживают различные интерфейсы связи, включая UART/USART, SPI, I2C, что позволяет легко интегрировать их в различные системы и взаимодействовать с другими устройствами.

Также многие модели семейства AVR имеют встроенные аналоговые компараторы, АЦП (аналогово-цифровые преобразователи) и ЦАП (цифро-аналоговые преобразователи), что позволяет обрабатывать аналоговые сигналы.

Семейство AVR делится на несколько подсемейств [2]:

1) TinyAVR:

– Flash-память до 16 Кбайт;
– RAM до 512 байт;
– ROM до 512 байт;
– Количество пинов ввода-вывода — от 4 до 18;
– Ограниченный набор периферийных устройств.

2) MegaAVR:

– Flash-память до 256 Кбайт;
– RAM до 16 Кбайт;
– ROM до 4 Кбайт;
– Количество пинов ввода-вывода — от 23 до 86;
– Расширенная система команд (поддержка ассемблера и языка C) и более широкий набор периферийных устройств.

3) XMEGA AVR:

– Flash-память до 384 Кбайт;
– RAM до 32 Кбайт;
– ROM до 4 Кбайт;
– Четырехканальный контроллер DMA для ускоренной работы с памятью и вводом/выводом.

Подсемейство MegaAVR подходит для решения поставленной задачи, так как оно предлагает большее количество пинов и объем памяти по

сравнению с TinyAVR, а также облегчает разработку программного кода, обеспечивая поддержку как ассемблера, так и языка C, на котором в настоящее время доступно огромное количество библиотек для работы с периферией. Подсемейство XMEGA AVR обладает избыточными ресурсами, поэтому данные микроконтроллеры можно отклонить.

Подсемейство MegaAVR предлагает на выбор широкую линейку микроконтроллеров, каждый из которых точно подходит под конкретный тип и масштаб задачи. В качестве такового был выбран ATmega16. Конкретный выбор обоснован следующими факторами.

ATmega16 имеет возможность настройки тактовой частоты вплоть до 16 МГц и способен выполнять достаточно сложные вычисления, необходимые для управления различными функциями беговой дорожки, такими как управление скоростью, мониторинг параметров и обработка пользовательского ввода в режиме реального времени.

Микроконтроллер имеет 32 порта ввода/вывода, что позволяет подключать различные датчики (например, датчики скорости, датчики расстояния) и устройства управления (например, моторы, дисплеи, кнопки). Это особенно важно для управления функциями беговой дорожки.

ATmega16 оснащен встроенным 10-битным АЦП, что позволяет легко считывать аналоговые сигналы, например, с датчиков пульса или других аналоговых устройств, используемых в беговых дорожках.

ATmega16 характеризуется низким энергопотреблением, что делает его подходящим для использования в устройствах, где важна эффективность и долговечность.

Микроконтроллеры AVR, включая ATmega16, известны своей надежностью и долговечностью, что делает их хорошим выбором для использования в устройствах, которые должны работать в течение длительного времени.

Структурная схема МК показана на рисунке 2 и УГО на рисунке 3 [2].

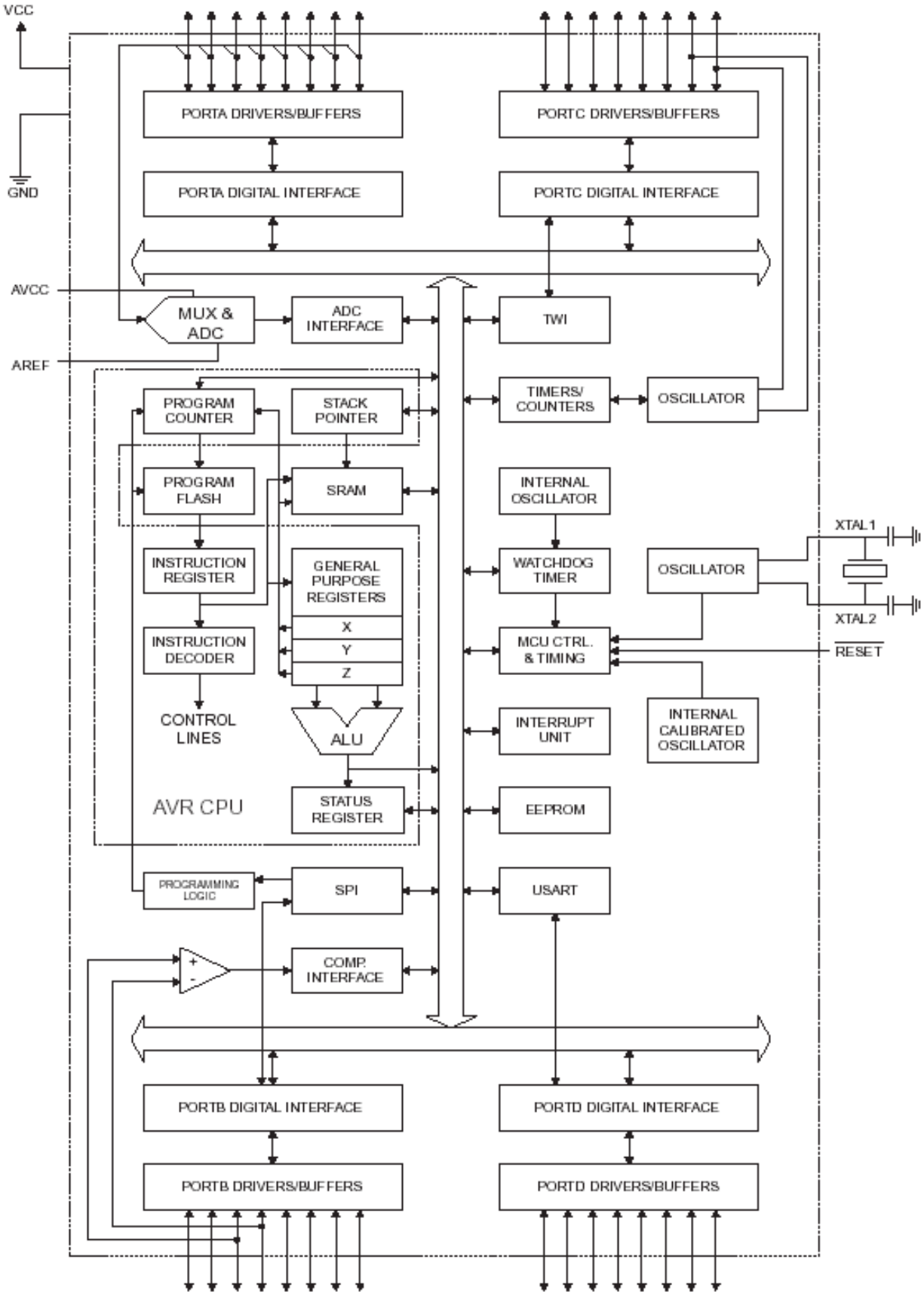

Рисунок 2 – Структурная схема МК ATmega16

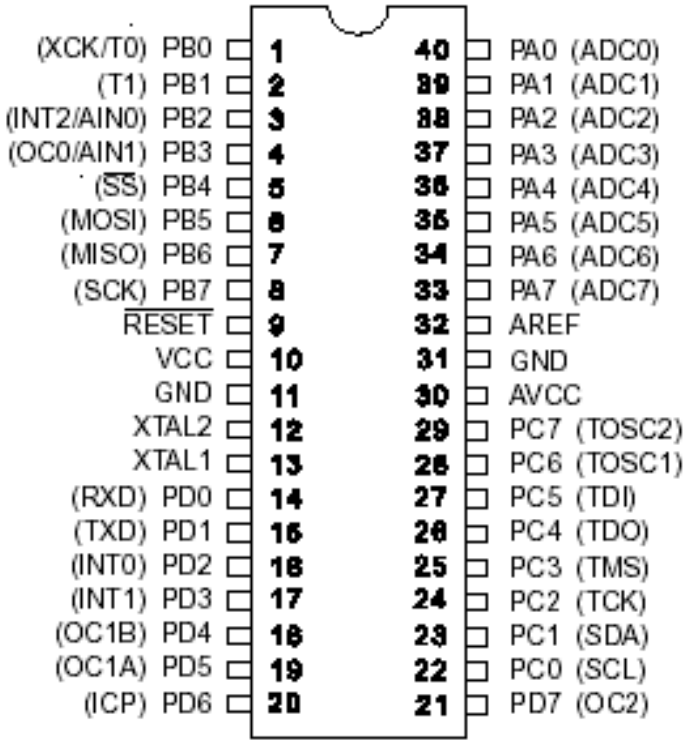

Рисунок 3 – УГО МК ATmega16

Он обладает следующими характеристиками [3]:

– 8-разрядный высокопроизводительный AVR микроконтроллер с малым потреблением

1. Прогрессивная RISC архитектура
   – 130 высокопроизводительных команд, большинство команд выполняется за один тактовый цикл;
   – 32 8-разрядных рабочих регистра общего назначения;
   – Полностью статическая работа;
   – Производительность приближается к 16 MIPS (при тактовой частоте 16 МГц);
   – Встроенный 2-цикловый перемножитель;
2. Энергонезависимая память программ и данных
   – 16 Кбайт внутрисистемно программируемой Flash памяти (In-System Self-Programmable Flash);
   – Обеспечивает 1000 циклов стирания/записи;

- Дополнительный сектор загрузочных кодов с независимыми битами блокировки;
- Внутрисистемное программирование встроенной программой загрузки;
- Обеспечен режим одновременного чтения/записи (Read-While-Write);
- 512 байт EEPROM;
- Обеспечивает 100000 циклов стирания/записи;
- 1 Кбайт встроенной SRAM;
- Программируемая блокировка, обеспечивающая защиту программных средств пользователя;

3. Встроенная периферия

- Два 8-разрядных таймера/счетчика с отдельным предварительным делителем, один с режимом сравнения;
- Один 16-разрядный таймер/счетчик с отдельным предварительным делителем и режимами захвата и сравнения;
- Счетчик реального времени с отдельным генератором;
- Четыре канала PWM;
- 8-канальный 10-разрядный аналого-цифровой преобразователь;
- Программируемый последовательный USART;
- Последовательный интерфейс SPI (ведущий/ведомый);
- Программируемый сторожевой таймер с отдельным встроенным генератором;
- Встроенный аналоговый компаратор;

4. Специальные микроконтроллерные функции

- Сброс по подаче питания и программируемый детектор кратковременного снижения напряжения питания;
- Встроенный калиброванный RC-генератор;
- Внутренние и внешние источники прерываний;
- Шесть режимов пониженного потребления: Idle, Power-save, Power-down, Standby, Extended Standby и снижения шумов ADC;

5. Выводы I/O и корпуса

– 32 программируемые линии ввода/вывода;

6. Рабочие напряжения

– 4,5 - 5,5 В;

7. Рабочая частота

– 0 - 16 МГц;

1.2.2 **Используемые элементы**

Для функционирования беговой дорожки в МК ATmega16 задействованы не все элементы его архитектуры. Выделим и опишем те, что используются во время функционирования схемы [4].

Порты A, B, C, D – назначения каждого из них описано в разделе *1.2.1.2*.

Указатель стека – используется для работы со стеком, при вызове подпрограмм. В коде они присутствуют.

SRAM – статическая память МК, где хранятся объявленные переменные.

Регистры общего назначения – предназначены для хранения операндов арифметико-логических операций, а также адресов или отдельных компонентов адресов ячеек памяти.

АЛУ – блок процессора, который под управлением устройства управления служит для выполнения арифметических и логических преобразований над данными.

SREG – регистр состояния, содержит набор флагов, показывающих текущее состояние работы микроконтроллера.

SPI – интерфейс для связи МК с другими внешними устройствами. В проекте используется только для прошивки МК.

Программный счетчик – используется для указания следующей команды выполняемой программы.

Память Flash – память МК, в котором хранится загруженная в него программа.

Регистры команды – содержит исполняемую в текущий момент (или следующий) команду, то есть команду, адресуемую счетчиком команд.

Блок расшифровки команд – выделяет код операции и операнды команды, а затем вызывает микропрограмму, которая исполняет данную команду.

Сигналы управления – синхронизируют обработку данных.

Логика программирования – устанавливает логику того, как программа будет вшита в МК.

Таймеры/счетчики – включает в себя 8-разрядные таймеры T0, T2 и 16-разрядный таймер T1. Во время работы данной программы используются таймеры T0 и T1:

таймер T0 считает время тренировки в секундах;

таймер T1 отвечает за формирование сигнала ШИМ, подаваемого на электродвигатель бегового полотна;

Управление синхронизацией и сбросом – в этом блоке обрабатываются тактовые сигналы и принимается сигнал сброса.

Прерывания – контроллер прерываний обрабатывает внешние прерывания и прерывания от периферийных устройств МК (таймеров, портов ввода/вывода). В данном проекте широко используется система прерываний. Обрабатываются прерывания при переполнении счетчика T0 и внешние прерывания от кнопок с панели управления.

UART – через этот интерфейс в МК передается информация из ПЭВМ. В регистр UART информация попадает через порт PD0 (RxD). Также с выхода PD1 (TxD) информация о тренировке подается на Bluetooth-модуль.

Генератор – генератор тактовых импульсов. Необходим для синхронизации работы МК.

1.2.3 Распределение портов

МК ATmega16 содержит четыре порта – A, B, C и D. Опишем назначение каждого из них.

Порт A: полностью выделен для передачи команд и данных на адресные входы LCD.

Порт B:

- PB0 – используется для индикации режима тренировки «Ручной»;
- PB1 – используется для индикации режима тренировки «Ходьба»;
- PB2 – используется для индикации режима тренировки «Бег»;
- PB4 – используется как управляющий сигнал RS для LCD; вход $\overline{SS}$, участвует при программировании МК;
- PB5 – используется как управляющий сигнал RW для LCD; вход MOSI, участвует при программировании МК;
- PB6 – используется как управляющий сигнал E для LCD; вход MISO, участвует при программировании МК;
- PB7 – вход SCK, участвует при программировании МК;

Порт C:

- PC0 – отвечает за принятие сигнала с кнопки «наклон +»;
- PC1 – отвечает за принятие сигнала с кнопки «наклон –»;
- PC3 – отвечает за принятие сигнала с кнопки «программа» и переключения текущей программы;
- PC4 – используется для управления шаговым двигателем наклона бегового полотна;
- PC5 – используется для управления шаговым двигателем наклона бегового полотна;
- PC6 – используется для управления шаговым двигателем наклона бегового полотна;
- PC7 – используется для управления шаговым двигателем наклона бегового полотна;

Порт D:

- PD0 – отвечает за вывод вводимой информации для проверки корректности ввода (RxD);

- PD1 – отвечает за принятие вводимой информации (TxD);
- PD2 – отвечает за принятие сигнала с кнопки «старт» и запуск устройства;
- PD3 – отвечает за принятие сигнала с кнопки «стоп» и остановку полотна;
- PD4 – отвечает за принятие сигнала с кнопки «скорость +»;
- PD5 – используется как выход сигнала ШИМ, поступающего на двигатель полотна и регулирующего скорость движения;
- PD6 – отвечает за принятие сигнала с кнопки «скорость –»;

1.2.4 Организация памяти

Схема организации памяти МК ATmega16 показана на рисунке 4.

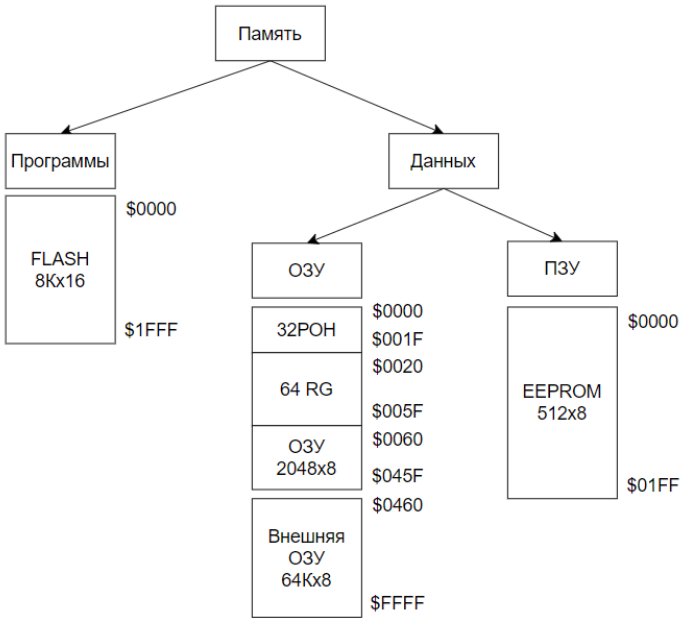

Рисунок 4 – Организация памяти МК ATmega16

Память программы предназначена для хранения последовательности команд, управляющих функционированием микроконтроллера, и имеет 16-ти битную организацию.

Все AVR имеют Flash-память программ, в выбранном МК её объем 8Кх16, то есть длина команды 16 разрядов. Информацию в flash-память можно мгновенно стереть и записать. В память микроконтроллера программа записывается с помощью программатора.

Память данных делится на три части:

1. Регистровая память – 32 регистра общего назначения и служебные регистры ввода/вывода.
2. Оперативная память (ОЗУ) – МК ATmega16 имеет объем внутреннего SRAM 1 Кбайт (с адреса $0060 до $045F). Число циклов чтения и записи в RAM не ограничено, но при отключении питания вся информация теряется. Также есть возможность подключить к МК ATmega16 внешнее ОЗУ объемом до 64Кбайт.
3. Энергонезависимая память (EEPROM) – эта память доступна МК в ходе выполнения, она предназначена для хранения промежуточных результатов. В выбранном МК ее объем составляет 512 байт. Также в неё могут быть загружены данные через программатор.

1.2.5 Прием данных от ПЭВМ

Приём данных от ПЭВМ происходит через драйвер MAX232. MAX232 – интегральная схема, преобразующая сигналы последовательного порта RS-232 в цифровые сигналы.

RS-232 – стандарт физического уровня для синхронного и асинхронного интерфейса (USART и UART). Обеспечивает передачу данных и некоторых специальных сигналов между терминалом и устройством приема. Сигнал, поступающий от интерфейса RS-232, через преобразователь передается в микроконтроллер на вход RxD.

К внешнему устройству MAX232 подключен через разъем DB9. Условно обозначен на схеме как XS1.

Внутреннее изображение MAX232 показано на рисунке 5. Назначение пинов описано в таблице 4 [5].

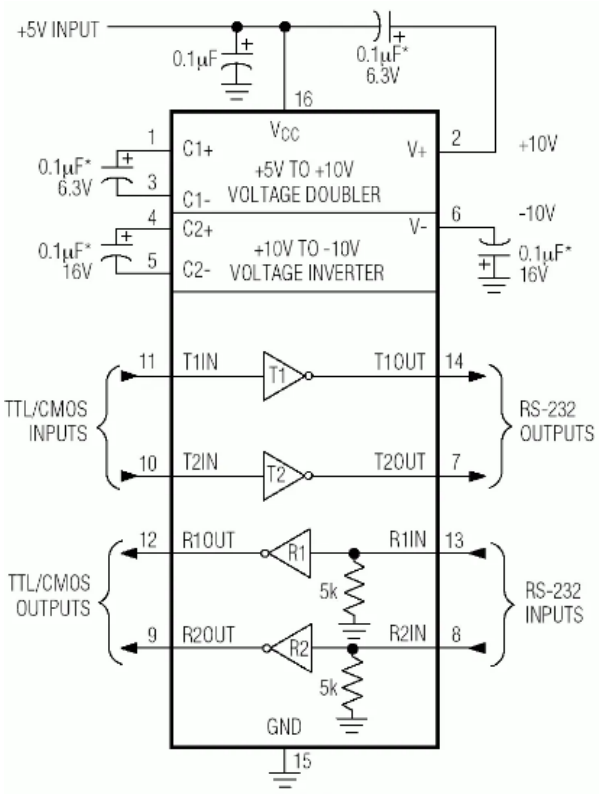

Рисунок 5 – Преобразователь MAX232

Таблица 2 - Назначение контактов MAX232

Номер	Имя	Тип	Описание
1	C1+	–	Положительный вывод C1 для подключения конденсатора
2	VS+	O	Выход положительного заряда для накопительного конденсатора
3	C1-	–	Отрицательный вывод C1 для подключения конденсатора
4	C2+	–	Положительный вывод C2 для подключения конденсатора
5	C2-	–	Отрицательный вывод C2 для подключения конденсатора
6	VS-	O	Выход отрицательного заряда для накопительного конденсатора
7, 14	T2OUT, T1OUT	O	Вывод данных по линии RS232
8, 13	R2IN, R1IN	I	Ввод данных по линии RS232

Продолжение таблицы 2

Номер	Имя	Тип	Описание
9, 12	R2OUT, R1OUT	O	Вывод логических данных
10, 11	T2IN, T1IN	I	Ввод логических данных
15	GND	–	Земля
16	Vcc	–	Напряжение питания, подключение к внешнему источнику питания 5 В

Когда микросхема MAX232 получает на вход логический "0" от внешнего устройства, она преобразует его в положительное напряжение от +5 до +15В, а когда получает логическую "1" - преобразует её в отрицательное напряжение от -5 до -15В. Обратные преобразования от RS-232 к внешнему устройству происходят аналогично.

1.2.6 Настройка канала передачи

Для передачи информации на Bluetooth-модуль в МК используется последовательный интерфейс USART. Для его работы необходимо настроить регистры управления UCSRA, UCSRB и UCSRC [6].

UCSRA. Биты регистра UCSRA показаны в таблице 3.

Таблица 3 – Биты регистра UCSRA

Номер	7	6	5	4	3	2	1	0
Название	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM
Доступ	R	R/W	R	R	R	R	R	R

Во время работы системы используются только биты RXC и UDRE [6].

RXC – флаг завершения приема. установлен, когда USART-модуль принимает данные в буфер. Используется при чтении данных из MAX232. Если бит RXC установлен, то в регистре UDR есть данные, которые нужно из него считать.

UDRE – флаг опустошения регистра данных и готовности регистра данных UDRE. Устанавливается в 1, когда буфер передатчика пуст и готов принимать новые данные. Используется при выводе данных из МК.

UCSRB. Биты регистра UCSRB показаны в таблице 4.

Таблица 4 – биты регистра UCSRB

Номер	7	6	5	4	3	2	1	0
Название	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
Доступ	R	R/W	R/W	R/W	R/W	R/W	R	R/W

Во время работы системы, для управления приемом и передачей информации, будут использоваться только биты TXEN и RXEN [6].

TXEN – разрешение передачи. Если разряд сбросится во время передачи, передача закончит свою работу до конца, и тогда передатчик выключится.

RXEN – при установке в 1 разрешается работа приемника. При сбросе разряда работа приемника запрещается, буфер сбрасывается.

UCSRC. Биты регистра UCSRC показаны в таблице 5.

Таблица 5 – Биты регистра UCSRC

Номер	7	6	5	4	3	2	1	0
Название	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Доступ	R	R/W	R/W	R/W	R/W	R/W	R	R/W

Во время работы системы будут использоваться только биты URSEL, UCSZ0 и UCSZ1 [6].

URSEL – регистры UCSRC и UBRRH находятся в одном адресном пространстве, и чтобы понять, куда записывать данные используется бит URSEL. При URSEL равным 1 в UCSRC, при 0 в UBRRH.

UCSZ0 и UCSZ1 – формат посылки. Каждый из них принимает значение 1 – для 8 битной посылки данных.

Скорость передачи определяется выражением [1]:

$$BAUD = \frac{f_{osc}}{16(UBRR + 1)}$$

где BAUD — скорость передачи (бод);

f_{osc} — тактовая частота микроконтроллера (Гц);

UBRR — содержимое регистров UBRR0H, UBRR0L контроллера скорости передачи.

Зададим скорость 9600 бод при $f_{osc} = 4$МГц. Получится:

$$UBRR = \frac{f_{osc}}{16 * BAUD} - 1 = \frac{4 * 10^6}{16 * 9600} - 1 = 25_{10}$$

UBRRL будет принимать значение 25.

1.2.7 Генератор тактовых импульсов и сброс

Для работы МК необходим тактовый генератор.

Внешний резонатор подключается к микроконтроллеру для обеспечения стабильной и точной работы его тактового генератора. Вот несколько причин, почему это важно:

1. Точность частоты: внешние резонаторы могут обеспечить более высокую точность и стабильность частоты по сравнению с встроенными генераторами. Это особенно важно в приложениях, где требуется точное время или синхронизация;
2. Настройка частоты: Внешние резонаторы позволяют выбрать нужную частоту тактирования для конкретного приложения. Это может быть полезно, если требуется изменить скорость работы микроконтроллера;
3. Устойчивость к температурным изменениям: Внешние резонаторы часто имеют лучшие характеристики стабильности частоты при изменении температуры, что делает их более надежными в различных условиях эксплуатации;
4. Снижение помех: Внешние резонаторы могут помочь уменьшить шум и помехи, которые могут влиять на работу микроконтроллера, что особенно важно в чувствительных приложениях;
5. Долговечность: Внешние резонаторы могут иметь более длительный срок службы по сравнению с встроенными генераторами, что может быть критично в некоторых приложениях;

Таким образом, использование внешнего резонатора может значительно улучшить производительность и надежность системы на основе микроконтроллера. Подключим к системе внешний кварцевый резонатор с частотой 4МГц – этого хватит для стабильной работы беговой дорожки.

В микроконтроллерах AVR, таких как ATmega, вывод $\overline{RESET}$ используется для инициализации устройства и его сброса. Когда питание включается, конденсатор разряжен, и $\overline{RESET}$ находится на уровне близком к нулю, что приводит к сбросу микроконтроллера. Затем конденсатор начинает заряжаться через резистор, и напряжение на $\overline{RESET}$ постепенно возрастает. Когда оно достигает порогового значения, микроконтроллер выходит из состояния сброса и начинает выполнение программы.

Время зарядки конденсатора определяет задержку перед запуском микроконтроллера, что позволяет обеспечить стабильность питания и минимизировать вероятность случайных сбросов.

Использование $\overline{RESET}$ также позволяет реализовать ручной сброс микроконтроллера, если на вывод $\overline{RESET}$ подать низкий уровень с помощью кнопки. Это может быть полезно для перезапуска программы или для инициализации устройства в определенном состоянии.

Подключим этот выход к питанию через резистор номиналом 10 кОМ и параллельно подключим конденсатор емкостью 100мкФ, а к нему землю.

1.2.8 Построение функциональной схемы

С учетом всех ранее подмеченных особенностей была спроектирована функциональная схема модели микроконтроллера беговой дорожки, показанная на рисунке 6 и в приложении Б [7, 8].

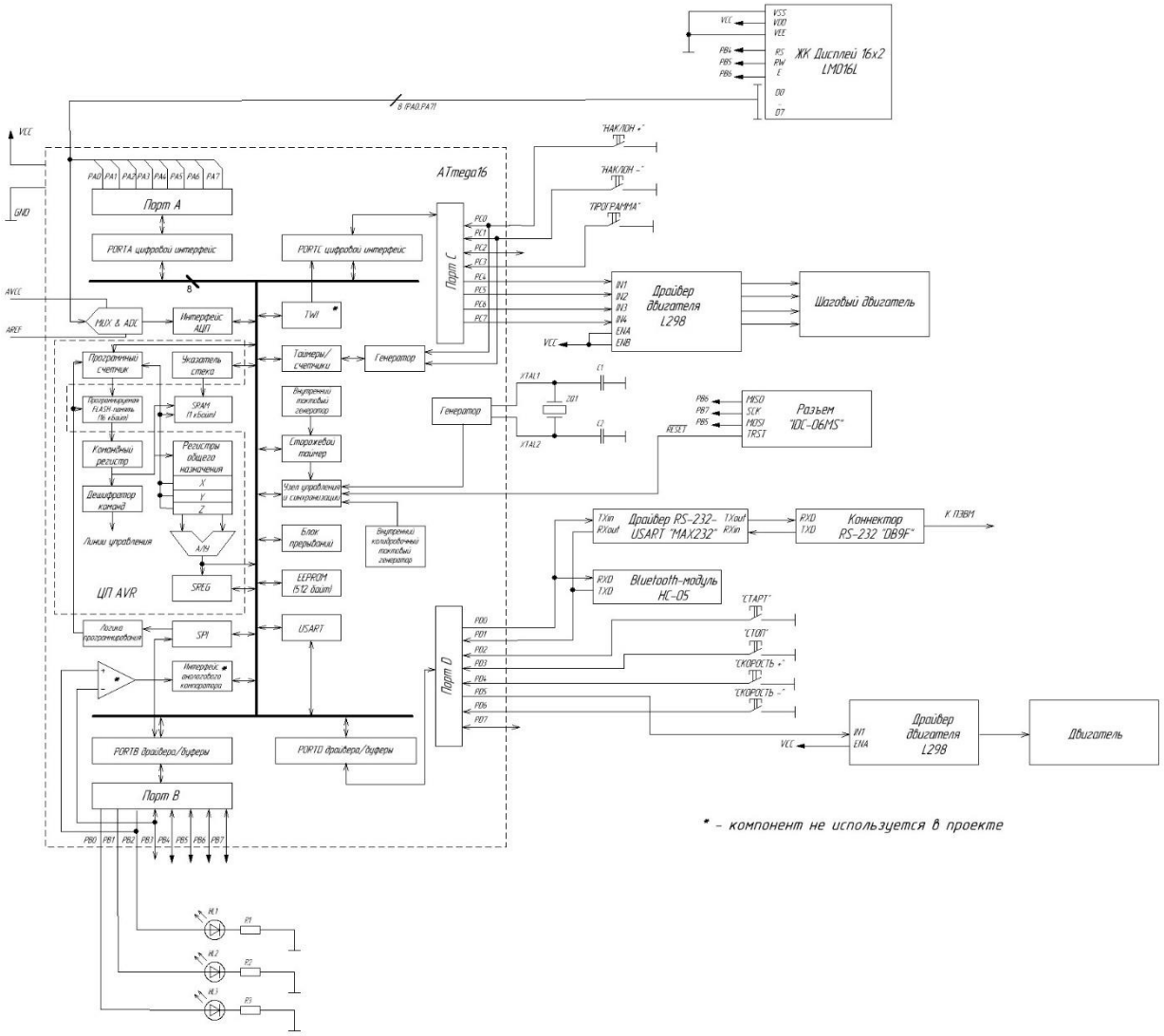

Рисунок 6 – Функциональная схема

1.3 Проектирование принципиальной схемы

1.3.1 Электродвигатель бегового полотна

Существуют 2 типа электродвигателей – постоянного тока (DC) и переменного тока (AC). Оба типа двигателей имеют свои особенности, преимущества и подходят для разных условий использования [9].

Двигатели постоянного тока и двигатели переменного тока имеют несколько ключевых различий, которые влияют на их применение, характеристики и конструкцию. Различия между ними представлены в таблице 6.

Таблица 6 – Сравнительная характеристика электродвигателей постоянного и переменного тока

Параметр	DC двигатель	AC двигатель
Принцип работы	Работает на постоянном токе	Работает на переменном токе
Контроль скорости	Легко регулируется изменением напряжения	Требует частотных преобразователей для регулирования скорости
Конструкция	Имеет щетки и коллектор	Обычно не имеет щеток (особенно асинхронные)
Эффективность	Меньшая эффективность на высоких мощностях, высокий крутящий момент при низких скоростях	Обычно более эффективен на высоких мощностях
Применение	Используется в точных приложениях (электромобили, игрушки)	Широко применяется в промышленных и бытовых устройствах (помпы, вентиляторы)

Проанализировав сравнительную характеристику, следует отдать предпочтение двигателю, работающему на постоянном токе. В качестве такового можно выбрать модель Baldor 42A. Его основные характеристики представлены в таблице 7.

Таблица 7 – Характеристики Baldor 42A

Параметр	Значение
Напряжение	24 В
Ток	10 А (максимальный рабочий ток)
Мощность	1/4 л.с. (примерно 186 Вт)
Крутящий момент	1.5 Н·м (примерно)
Скорость	1750 об/мин (примерно)
Сопротивление обмотки	Зависит от температуры и конструкции (обычно в пределах 1-2 Ом)
Эффективность	Обычно около 70-85% (в зависимости от нагрузки)
Рабочая температура	-20°C до +40°C

1.3.2 Расчет скорости электродвигателя

Чтобы рассчитать максимальную скорость в километрах в час для двигателя Baldor 42A, нужно знать его максимальные обороты в минуту и диаметр ролика бегового полотна, к которому он подключен.

В большинстве современных беговых дорожек диаметр ролика равен 10 см. Это хорошее среднее значение, так как чем больше диаметр ролика, тем меньшая нагрузка пойдет на двигатель. Значение диаметра D = 10 см можно считать оптимальным.

Двигатель Baldor 42A обычно имеет максимальную скорость оборотов RPM около 1750 об/мин.

Положим, что диаметр ролика составляет 10 см, тогда рассчитаем длину окружности колеса по формуле:

$$L = 2\pi R = \pi D = 3.14 * 0.1 \approx 0.314\text{м} \quad (1)$$

А скорость найдем по формуле:

$$v \text{ км/} = RPM * 0.314 * \frac{60}{1000} = 1750 * 0.314 * \frac{60}{1000} \approx 32.8 \text{ км/ч} \quad (2)$$

Нетренированным людям с трудом удастся достичь скорости 15 км/ч, но они не смогут поддерживать её более одного километра. При беге на дальние и средние дистанции обычная скорость – 10-12 км/ч. Чаще всего для домашних дорожек она не превышает 16 км/ч, а для профессиональных моделей — 25 км/ч и выше. На такой скорости бег становится очень интенсивным и подходит только для опытных спортсменов [10].

Пусть максимальная скорость беговой дорожки будет равна 20 км/ч. Она понадобится в будущем при расчете текущей нагрузки и подсчета калорий.

1.3.3 Шаговый двигатель для наклона полотна

Шаговый двигатель — это тип электродвигателя, который делит один полный оборот на определённое количество равных шагов [11]. Он работает по принципу, при котором электрические импульсы подаются на обмотки двигателя, что заставляет его вращаться на фиксированный угол. Это позволяет точно контролировать положение и скорость вращения.

Следует учесть, что двигатель должен справиться с нагрузкой. Под нагрузкой понимаем вес пользователя. Положим средний вес пользователя равным 75 кг. Тогда выберем в качестве такого двигателя модель NEMA 23. Его характеристики приведены в таблице 8.

Таблица 8 – Характеристика шагового двигателя NEMA 23

Параметр	Значение
Рабочее напряжение	12-48 В (в зависимости от модели)
Номинальный ток	1.0 - 3.0 А (в зависимости от модели)
Крутящий момент	1.0 - 2.0 Нм (или выше)
Шаг (угол)	1.8° (200 шагов на полный оборот)
Сопротивление обмотки	1.2 - 3.5 Ом (в зависимости от модели)
Индуктивность	2.0 - 5.0 мГн (в зависимости от модели)
Максимальная скорость	1000 - 3000 шагов/с (в зависимости от драйвера и нагрузки)
Размеры	57.15 x 57.15 x 112.0 мм (обычные размеры)

Этой модели шагового двигателя будет достаточно, чтобы поднять беговое полотно с пользователем весом в 75 кг, но потребуется редуктор для увеличения крутящего момента. Это выходит за рамки данной работы.

1.3.4 Выбор драйверов для двигателей

Выбор драйвера для двигателя зависит от нескольких факторов, таких как [12]:

- тип двигателя (шаговый, постоянного/переменного тока);
- напряжение питания;
- максимальный ток;
- режим управления (драйвер должен поддерживать управление по сигналу ШИМ);
- количество каналов;

– дополнительные функции (защита от перегрузки, реверс вращения и другие).

Выберем популярный в настоящее время драйвер L298, характеристики которого представлены в таблице 9.

Таблица 9 – Характеристики драйвера L298

Параметр	Значение
Рабочее напряжение (Vs)	5 В до 46 В
Максимальное напряжение (Vmax)	50 В (не рекомендуется превышать)
Максимальный выходной ток	2 А на канал
Пиковый ток	до 3 А на канал (кратковременно)
Максимальная рассеиваемая мощность	25 Вт (при 25°C с теплоотводом)
Рабочая температура	-25°C до +130°C
Температура хранения	-40°C до +150°C
Управление	TTL, CMOS (логические уровни)
Поддержка ШИМ	Да
Входы для управления	IN1, IN2 (канал 1); IN3, IN4 (канал 2)
Выходы	OUT1, OUT2 (канал 1); OUT3, OUT4 (канал 2)
Защита	Защита от перегрева, перенапряжения и короткого замыкания (внешняя защита рекомендуется)

Изучив характеристику драйвера, можно утверждать, что он подойдет для управления как двигателем бегового полотна Baldor 42A, так и шаговым двигателем NEMA 23. На принципиальной схеме понадобится два таких драйвера, так как управление шаговым двигателем занимает 2 канала.

1.3.5 Настройка таймеров

В данной работе активно используется работа с системой прерываний. Прерывания генерируются с помощью таймеров. Каждый таймер отвечает за конкретную задачу.

Таймер T0 отвечает за подсчет времени тренировки в секундах. Его нужно настроить на прерывания каждую секунду. Для настройки используется регистр TCCR0, представленный на рисунке 7 [13].

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Рисунок 7 – Регистр TCCR0

Биты WGMxx используются для выбора режима таймера. При установке 1 в бит WGM01 устанавливается режим CTC (Clear Timer on Compare Match), который позволяет таймеру сбрасываться при достижении значения, заданного в регистре OCR0. Это позволяет таймеру генерировать прерывание, когда он достигает определенного значения.

Биты COMxx задают поведение пина OC0. Он не используются, поэтому они равны 0.

Биты CSxx отвечают за предделитель. С их помощью можно также остановить работу таймера, если записать в них нули.

Пусть частота работы микроконтроллера равна 4МГц. Тогда с предделителем равным 256 частота его работы составит:

$$\nu_{prescaler} = \frac{\nu}{256} = \frac{4\text{МГц}}{256} = 15625 \text{ тактов}$$

Требуется 15625 тактов для отсчета одной секунды. В регистр OCR0 запишем 255. Рассчитаем количество переполнений, необходимое для отсчета 1 секунды.

$$n = \frac{15625}{255} \approx 61 \text{ переполнение}$$

Настраиваем таймер с предделителем 256 и режимом CTC. Значение для OCR0 устанавливается на 255, чтобы таймер переполнялся каждый раз, когда он достигает этого значения. Разрешаем прерывание по совпадению, чтобы увеличить счетчик переполнений каждый раз, когда таймер переполняется. В основном цикле проверяем, прошло ли 61 переполнение (что соответствует 1 секунде), и выполняем необходимые действия.

Таким образом, эти настройки позволят корректно отмерять одну секунду времени при частоте 4 МГц.

Для таймера T1 основной задачей является формирование сигнала ШИМ, который будет регулировать скорость беговой дорожки.

Для его настройки используются два регистра: TCCR1A и TCCR1B представленные на рисунках 8 и 9 соответственно.

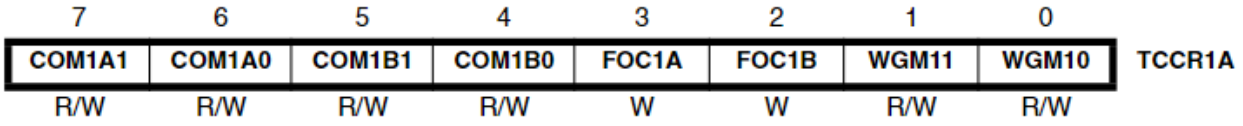

Рисунок 8 – регистр TCCR1A

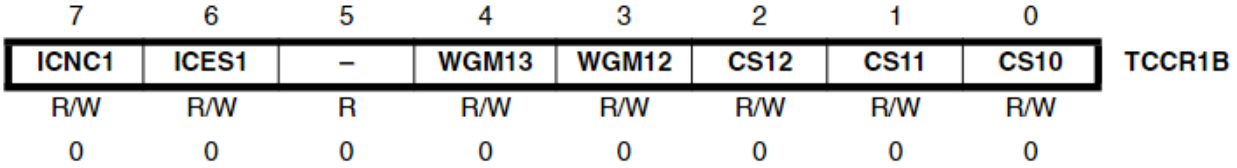

Рисунок 9 – регистр TCCR1B

Биты с аналогичными названиями сохраняют свою функциональность. WGM11 устанавливает бит, который включает режим работы таймера (режим Fast PWM).

COM1A1 устанавливает бит, который включает режим сравнения для выхода OC1A. Это означает, что выход OC1A будет установлен в высокое состояние при совпадении и очищен при переполнении.

WGM12 и WGM13 устанавливают биты, в комбинации с WGM11 они настраивают таймер на режим Fast PWM.

Регулировка скорости происходит путем записи значений в регистр OCR1A. Максимальная скорость беговой дорожки была принята равной 20 км/ч. Разобьем отрезок от 0 до 255 на шаги через каждые 20 пунктов. Получим набор скоростей беговой дорожки, представленный в таблице 10.

Таблица 10 – Режимы скорости беговой дорожки

№ Режима	Значение OCR1A	% от максимальной скорости	Скорость в км/ч
0	0	0	0
1	20	7,84%	1,57

Продолжение таблицы 10

№ Режима	Значение OCR1A	% От максимальной скорости	Скорость в км/ч
2	40	15,69%	3,14
3	60	23,53%	4,71
4	80	31,37%	6,27
5	100	39,22%	7,84
6	120	47,06%	9,41
7	140	54,90%	10,98
8	160	62,75%	12,55
9	180	70,59%	14,12
10	200	78,43%	15,69
11	220	86,27%	17,25
12	240	94,12%	18,82

Таймер T2 в свою очередь отвечает за опрос кнопок панели управления. Обоснуем этот выбор реализации.

Существуют блокирующие и неблокирующие способы опроса нажатия кнопок. Блокирующие варианты останавливают выполнение основного цикла программы, пока не обработают прерывание с кнопки, что может быть критично в случае беговой дорожки. Также это может повлиять на загрузку и переполнение стека в ходе работы программы, из-за чего микроконтроллер может не справиться с вычислениями [14].

Будет логичнее реализовать опрос кнопок через определенный интервал времени. Легче всего это сделать с помощью таймера.

Настройка таймера T2 выполняется с помощью регистров TCCR2, представленном на рисунке 10, и TCNT [13].

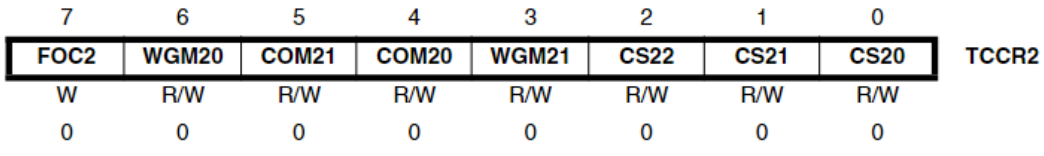

Рисунок 10 – Регистр TCCR2

Оптимальное время для опроса кнопок равно 500мс, так как реакция человека медленная и нужно, чтобы он успевал реагировать на произведённое действие. Частота – это количество тактов в секунду. Следовательно, за половину секунды пройдет лишь половина тактов рабочей частоты 4МГц, то есть 2 миллиона тактов.

Если взять предделитель равным 64, то для 500мс понадобится:

$$\frac{2000000}{64} = 31250 \text{ тактов}$$

Таймер T2 8-разрядный. Максимальное значение в регистре TCNT равно 255. Тогда получим количество переполнений для подсчета:

$$n = \frac{31250}{256} \approx 123 \text{ переполнения}$$

Чтобы получить ровно 31250 тактов, необходимо посчитать начальное значение, которое нужно записать в TCNT. Оно вычисляется следующим образом:

$$256 - \left(\frac{31250}{123}\right) \approx 256 - 254.07 \approx 2$$

Таймер будет переполняться и вызывать прерывание с опросом кнопок каждые 500мс.

Установка предделителя, равного 64, определяется установкой в 1 битов CS22 и CS21 регистра TCCR2. Разрешение на прерывания определяется установкой в 1 бита TOIE2 в регистре TIMSK.

1.3.6 Выбор устройства отображения данных

Устройство отображения данных должно предоставить пользователю информацию о текущей тренировке, а именно:

- текущая скорость;
- наклон бегового полотна;
- прошедшее время тренировки;
- сожженные калории;
- выбранная программа.

В качестве такого устройства был выбран жидкокристаллический дисплей LCD LM016L, характеристики которого представлены в таблице 11 [15].

Таблица 11 – Характеристики ЖК-дисплея LM016L

Параметр	Значение
Напряжение питания	4.5 - 5.5 В
Ток потребления	15 - 25 мА
Потребляемая мощность	75 - 137.5 мВт
Входное сопротивление	10 кОм
Входной ток	1 мА
Напряжение логического уровня	2.4 - 5.5 В
Частота тактового сигнала	1 - 4 МГц
Время доступа	100 - 200 нс

Для отображения информации на данном LCD отведено лишь 2 строки по 16 символов. Управление LCD-дисплеем LM016L осуществляется с помощью нескольких сигналов, которые обеспечивают передачу данных и команд. Вот краткое описание основных сигналов управления:

- RS (Register Select) - определяет, будет ли передаваемая информация интерпретироваться как команда (RS = 0) или как данные (RS = 1).
- RW (Read/Write) - указывает направление передачи данных. Если RW = 0, данные записываются в дисплей (операция записи). Если RW = 1, данные считываются из дисплея (операция чтения).
- E (Enable) - активирует передачу данных. При переходе сигнала E из низкого в высокий уровень (положительный фронт) данные, находящиеся на шинах данных, будут считаны или записаны.
- Данные (D0-D7) - 8-битная шина данных, по которой передаются команды и данные. Дисплей может работать в 4-битном или 8-битном режиме. В 4-битном режиме используются только D4-D7.

1.3.7 Разъем программатора

Для подключения программатора необходим специальный разъем IDC-06MS. Подключение осуществляется при помощи интерфейса SPI. Под него

на МК ATmega16 задействованы последние 4 контакта порта PB. На принципиальной схеме, которая показана в разделе *1.3.5*, условное обозначение – XP3.

Он имеет следующие разъемы для подключения к МК [16]:

- MISO (Master Input Slave Output) – для передачи данных от микроконтроллера в программатор;
- SCK – тактовый сигнал;
- MOSI (Master Output Slave Input) – для передачи данных от программатора в микроконтроллер;
- $\overline{RESET}$ – сигналом на $\overline{RESET}$ программатор вводит контроллер в режим программирования.

1.3.8 Подключение цепи питания

Разрабатываемой микроконтроллерной системе необходимо питание. В данной работе потребуется 3 блока питания: на 5В, 12В и 24В для всей схемы, шагового двигателя и двигателя бегового полотна соответственно. В качестве таких элементов можно выбрать ARDV-15-5B, LPV-35-12 и ELG-240-24A. Их технические характеристики представлены в таблице 12:

Таблица 12 – Технические характеристики блоков питания

	Выходной ток, А	Выходное напряжение, В	Входное напряжение, В	Пусковой ток, А
ARDV-15-5B	3	5	100-240	40
LPV-35-12	3	12	90-264	55
ELG-240-24A	10	24	142-431	60

Питание подключим через разъемы DS-201, которые обозначены на схеме как XS1, XS2 и XS3.

1.3.9 Расчет потребляемой мощности

Потребляемая мощность – это мощность, потребляемая интегральной схемой, которая работает в заданном режиме соответствующего источника питания.

Чтобы рассчитать суммарную мощность, рассчитаем мощность каждого элемента. На все микросхемы подается напряжение +5В. Мощность, потребляемая один устройством, в статическом режиме, рассчитывается формулой:

$$P = U * I$$

где U – напряжение питания (В);

I – ток потребления микросхемы (мА).

Также в схеме присутствуют резисторы. Одно из предназначений резисторов на схеме – уменьшить значение тока, подаваемого на устройство. В проекте беговой дорожки установлены подтягивающие резисторы у кнопок.

Для подтягивающих резисторов используем формулу (1):

$$P_{R_i} = \frac{U^2}{R_i} \quad (1)$$

$$P = \frac{5^2\ \text{В}}{10 * 10^3\ \text{Ом}} = 2{,}5\text{мВт}$$

Таких резисторов на схеме размещено 9, поэтому их суммарная потребляемая мощность будет равна 22,5мВт.

На схеме также должны быть расположены резисторы, подключенные последовательно к светодиодам и ограничивающие ток. Чтобы рассчитать их номинал и, следовательно, потребляемую мощность, необходимо узнать падение напряжения. Выберем модель светодиода L-7113ID-5V. Его максимальное рабочее напряжение составляет 2,5В. Тогда падение напряжения вычислим по формуле:

$$U_R = U_{source} - U_f = 5\text{В} - 2.5\text{В} = 2.5\text{В}$$

По закону Ома найдем сопротивление резисторов:

$$R = \frac{U_\text{в}}{I} = \frac{2.5\text{В}}{0.02\text{А}} = 125\ \text{Ом}$$

Данного номинала в таблице е24 не существует. Возьмем резисторы МЛТ-2 номиналом 130 Ом. Рассчитаем потребляемую мощность для этих резисторов.

$$P = I^2 * R = 0{,}02^2 A * 130 * 3\text{шт} = 0{,}156\text{Вт} \approx\ 150\text{мВт}$$

Теперь для самих светодиодов.

$$P = U_f * I = 2.5\text{В} * 0.02\text{А} * 3\text{шт} = 0{,}15\text{Вт} = 150\text{мВт}$$

Расчет потребляемого напряжения для каждой микросхемы проекта показан в таблице 13.

Таблица 13 – Потребляемая микросхемами мощность

Микросхема	Ток потребления, мА	Потребляемая мощность, мВт	Количество устройств	Суммарная потребляемая мощность, мВт
ATmega16	25	125	1	125
MAX232	8	40	1	40
L298 (Baldor 42A)	2000	10000	1	10000
Канал L298 (NEMA 23)	2000	10000	2	20000
Baldor 42	10000	240000	1	240000
NEMA 23	3000	36000	1	36000
LM016L	20	100	1	100

$$P_\Sigma = P_{ATmega16} + P_{MAX232} + P_{L298-Baldor42A} + P_{L298-NEMA23} + P_{Baldor42A} + P_{NEMA23} + P_R + P_{HL} = 125 + 40 + 10000 + 20000 + 240000 + 36000 + 100 + 150 + 22.5 + 150 = 306587.5\text{мВт} = 306.6\text{Вт}$$

Суммарная потребляемая мощность системы равна 306.6 Вт.

1.3.10 Построение принципиальной схемы

На основе всех вышеописанных сведений была спроектирована принципиальная схема разрабатываемой системы, показанная на рисунке 11 и в приложении Б [7, 8].

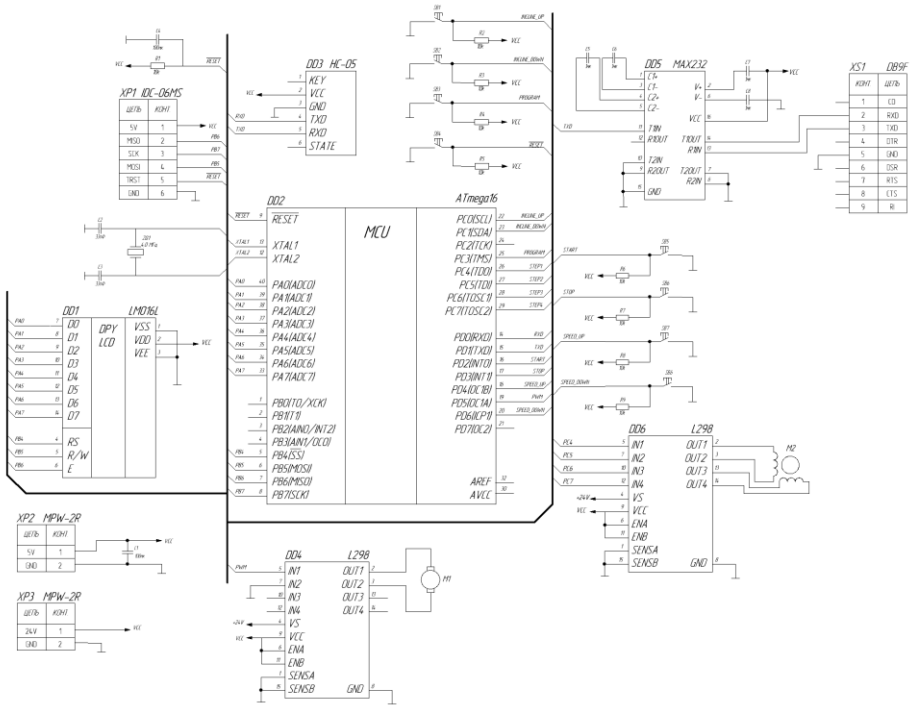

Рисунок 11 – Принципиальная схема

1.4 Алгоритмы работы системы

1.4.1 Функция Main

Для модели беговой дорожки была разработана программа, схема алгоритма которой представлена на рисунке 11.

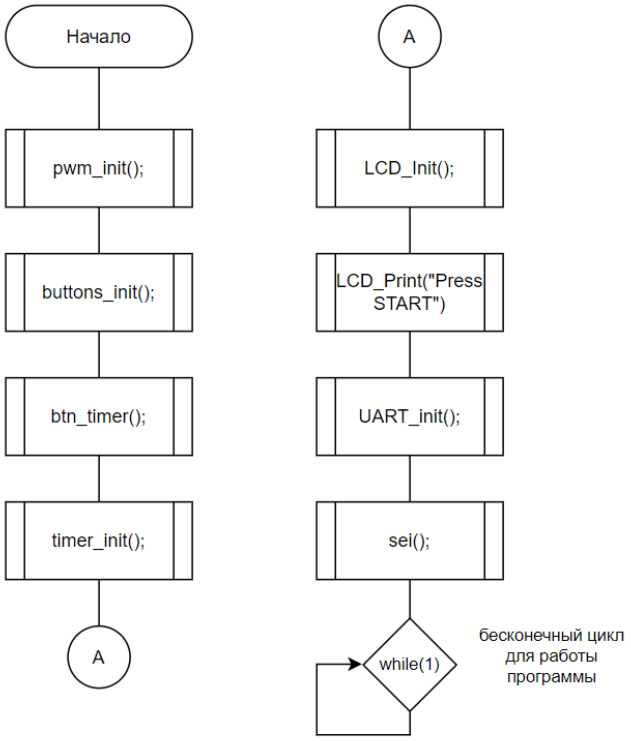

Рисунок 11 – Схема алгоритма программы

Работа начинается с функции main, из которой вызываются все остальные функции.

Сначала идет вызов функции pwm_init, которая задает настройки ШИМ и пин для двигателя, вращающего беговое полотно, и настройки пинов для управления шаговым двигателем.

Далее идет вызов функции buttons_init, которая настраивает пины кнопок на вход и включает подтягивающие резисторы.

Следующим идет вызов функции btn_timer_init, которая настраивает таймер T2 на прерывания, в которых происходит опрос кнопок.

Дальше происходит вызов функции timer_init. В ней устанавливаются настройки таймера T0, который будет отсчитывать время в секундах и обновлять информацию на дисплее.

Далее выполняется настройка работы дисплея в фукнции LCD_Init, выводится приветственное сообщение.

Затем настраивается канал передачи данных в функции USART_init и разрешаются глобальные прерывания командой sei.

Запускается бесконечный цикл, но он пуст, так все функции перенесены в прерывания.

1.4.2 Используемые при работе подпрограммы

В разработанной программе используется несколько процедур и функций, каждая из которых решает свою задачу. Рассмотрим их подробнее.

1. Процедуры для LCD

Данные процедуры выполняют инициализацию и настройку LCD для корректного отображения данных о тренировке, посылают команды и данные на порт данных, выводят сообщения на экран и обновляют информацию о тренировке. Их схемы алгоритмов представлены на рисунке 12.

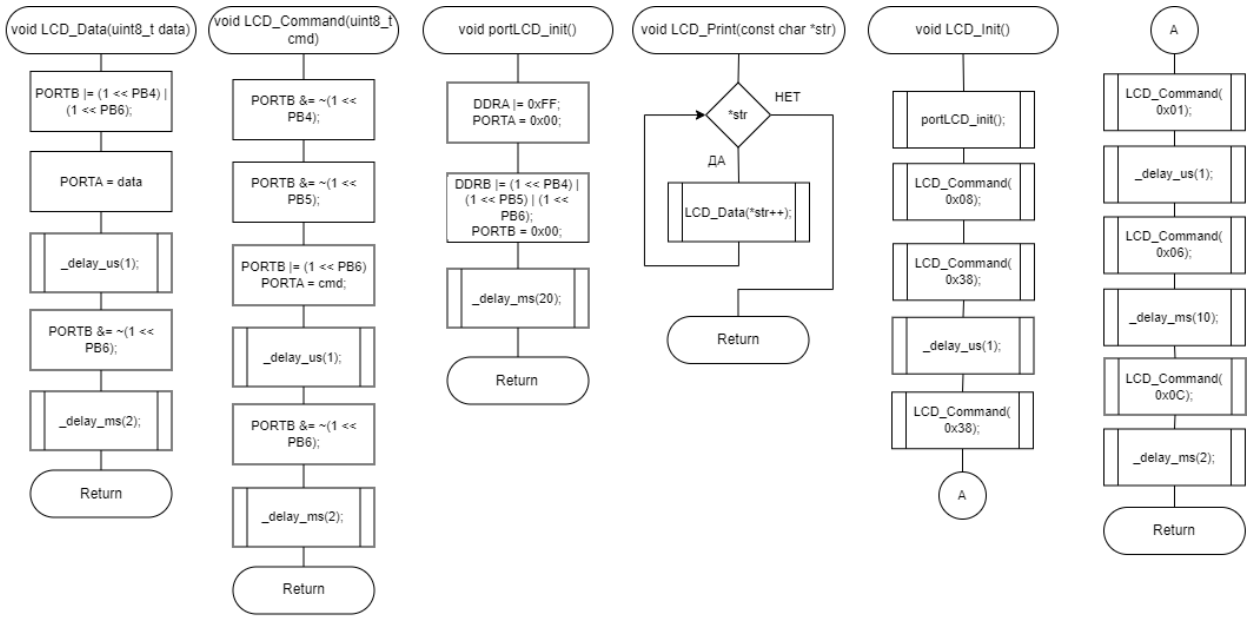

Рисунок 12 – Схемы алгоритмов подпрограмм LCD

2. Процедуры USART

Главное предназначение этих процедур – обеспечить корректную передачу по беспроводному Bluetooth-модулю. Схемы алгоритмов представлены на рисунке 13.

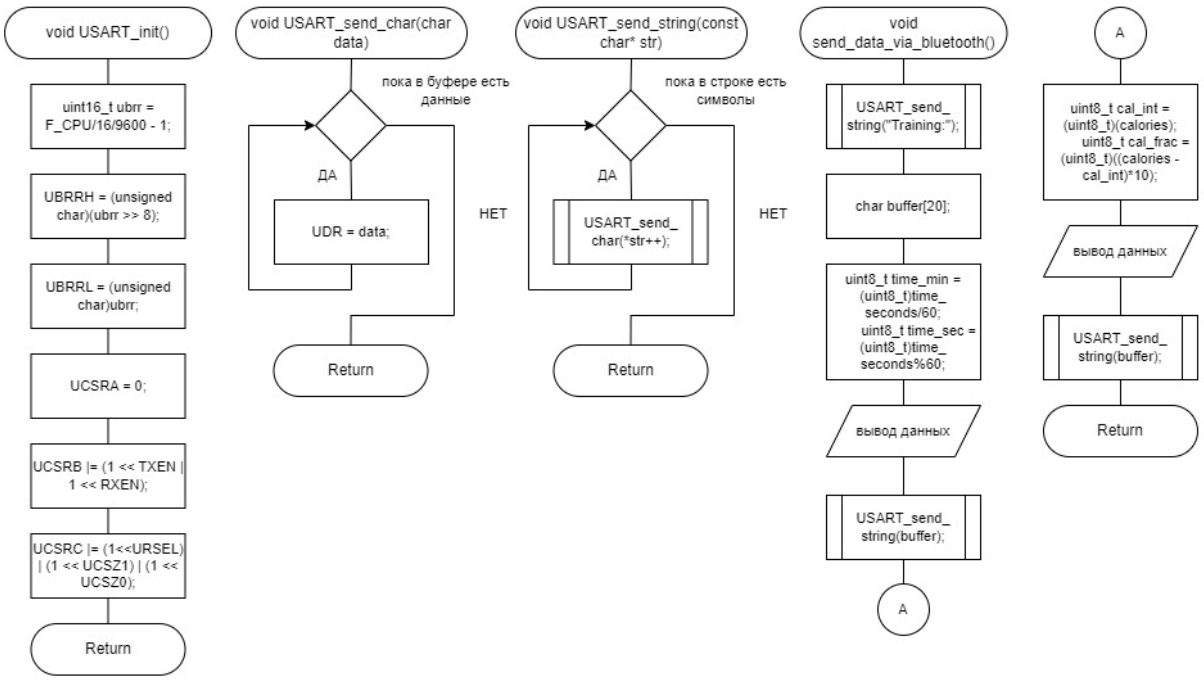

Рисунок 13 – Схемы алгоритмов подпрограмм USART

3. Подпрограмма подсчета калорий

Данная процедура объединена вместе с прерываниями таймера времени для оптимизации работы программы, так как часто обновлять данные о калориях, а значит, добавлять дополнительную нагрузку на ЦП микроконтроллера, не имеет смысла. Расчет калорий ведется по следующим формулам [17].

Если скорость движения меньше 100м/с:

$$Cal = (0.1 * v) + (1.8 * v * grade) + 3.5,$$

где cal – искомое количество калорий,

v – скорость в метрах в минуту,

$grade$ – угол наклона в процентном отношении.

Если скорость движения больше 100м/с:

$$Cal = (0.2 * v) + (0.9 * v * grade) + 3.5,$$

где cal – искомое количество калорий,

v – скорость в метрах в минуту,

$grade$ – угол наклона в процентном отношении.

Схема алгоритма этой процедуры представлена на рисунке 14.

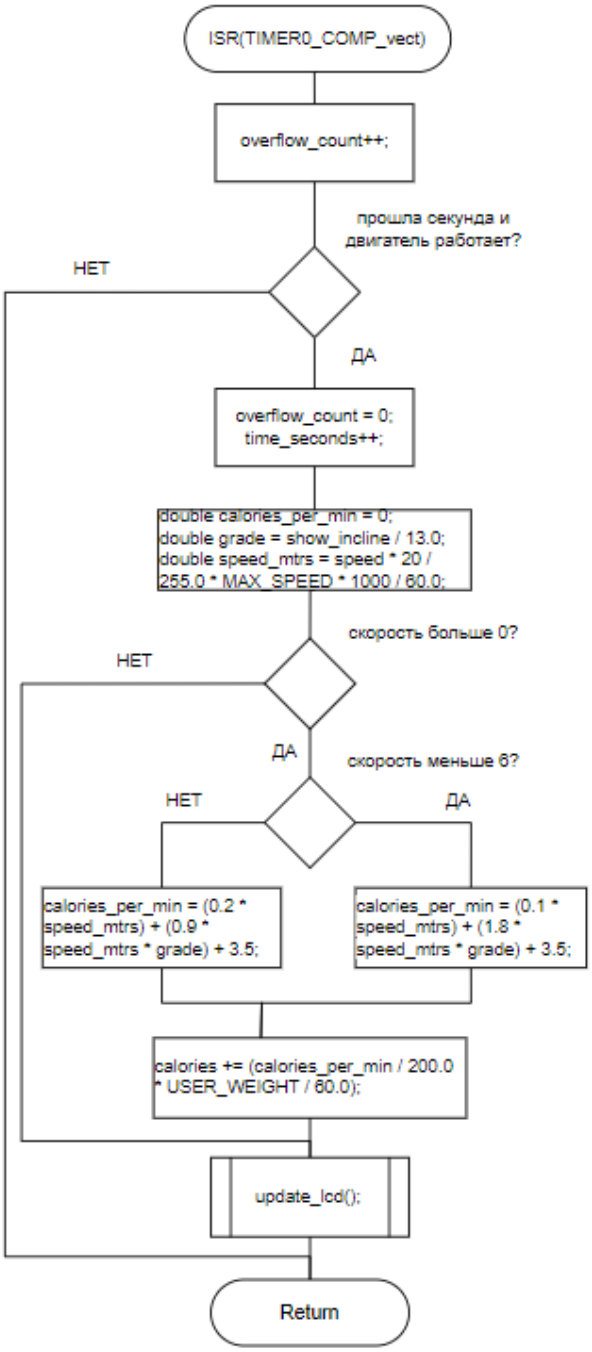

Рисунок 14 – Схема алгоритма подпрограммы подсчета калорий и таймера времени

4. Процедуры настройки таймеров, ШИМ и кнопок на ввод

Процедуры pwm_init(), buttons_init(), btn_timer_init() и timer_init() устанавливают начальные настройки для сигнала ШИМ, который подается на двигатель, кнопок панели управления, таймеров опроса кнопок и подсчета времени в секундах соответственно. Их схемы алгоритма представлены на рисунке 15.

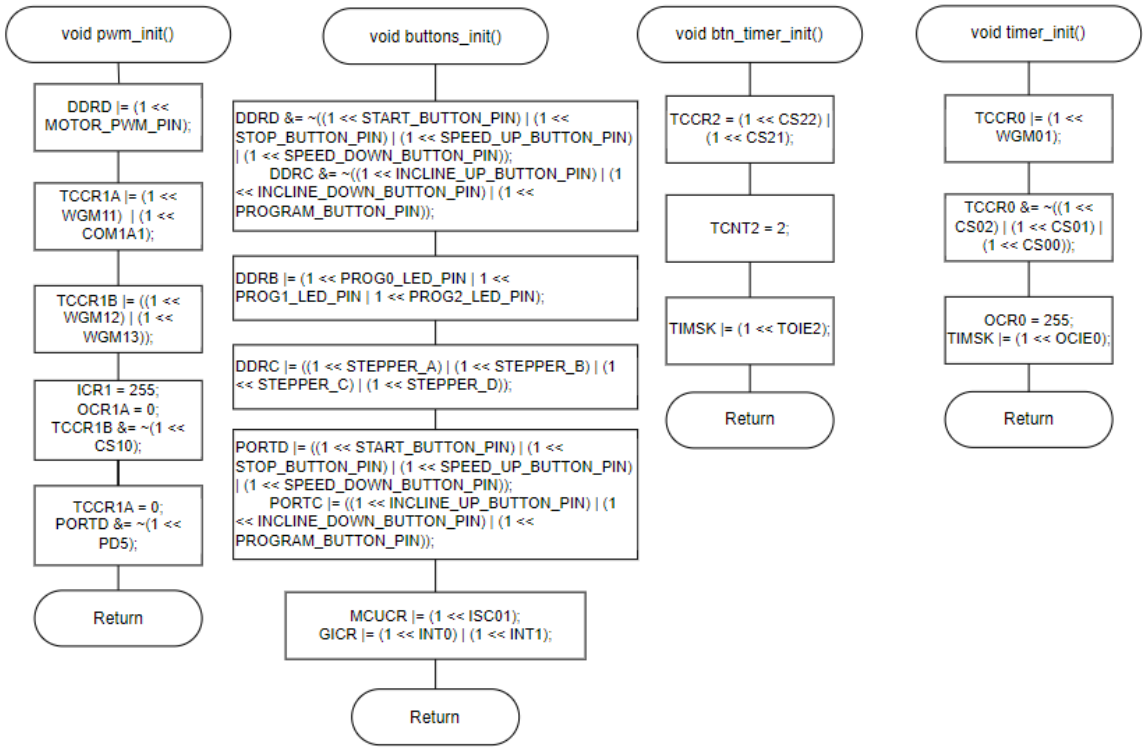

Рисунок 15 – Схемы алгоритмов настройки таймеров, кнопок и ШИМ

5. Управление шаговым двигателем

Управление шаговым двигателем реализовано с помощью двух процедур – incline_up() и incline_down() – и предполагает собой подачу электрических импульсов на обмотку двигателя. Импульсы подаются с пинов PC4, PC5, PC6 и PC7. Схемы алгоритмов этих процедур представлены на рисунках 16-17.

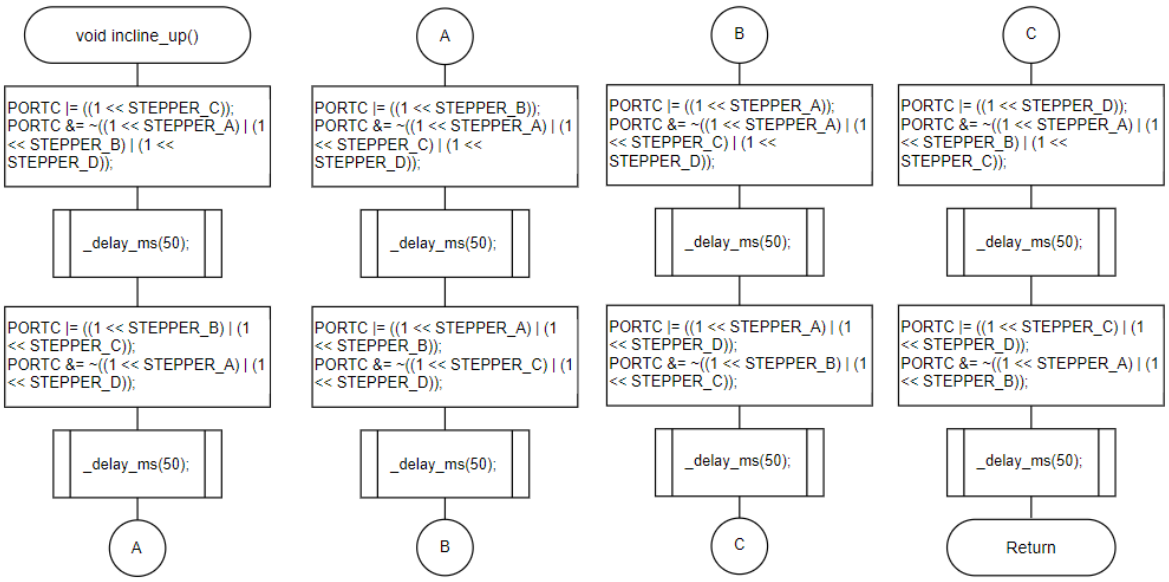

Рисунок 16 – Схема алгоритма подъема шагового двигателя

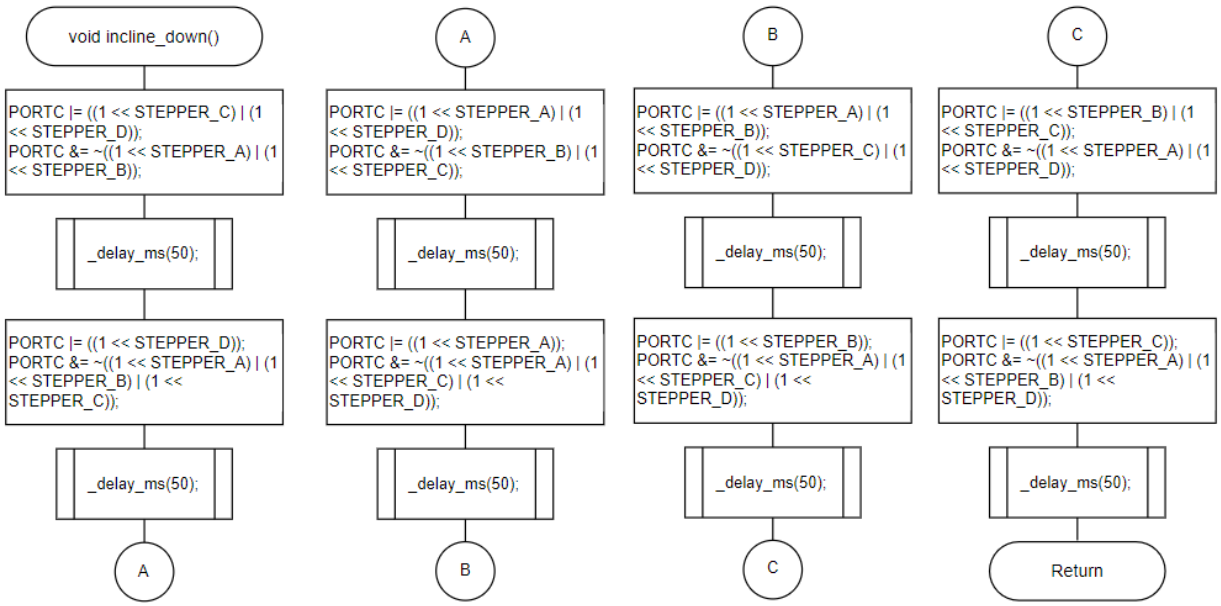

Рисунок 17 – Схема алгоритма спуска шагового двигателя

6. Процедура опроса кнопок

Опрос кнопок, как и было сказано ранее, реализован с помощью прерывания таймера T2. Это снижает нагрузку на стек и повышает производительность системы при вызове других процедур. Схема алгоритма этой процедуры представлена на рисунке 18.

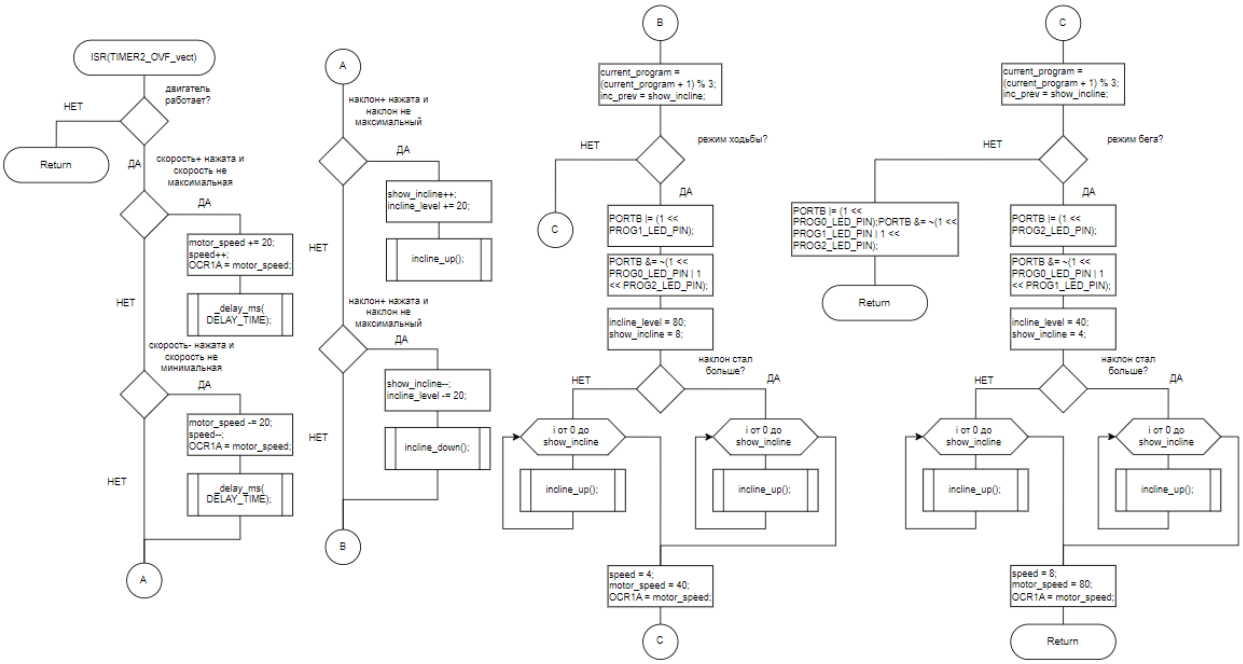

Рисунок 18 – Схема алгоритма подпрограммы обработки нажатия кнопок

7. Прерывания кнопок «старт» и «стоп»

Эти кнопки являются самыми важными во всей системе, так как они запускают и выключают устройство по требованию пользователя. Кнопка «стоп» также совмещает в себе функцию «пауза». При первом нажатии на нее данные о текущей тренировке не удаляются, а работа беговой дорожки приостанавливается с возможностью продолжить тренировку нажатием на кнопку «старт». При повторном нажатии на кнопку «стоп», тренировка заканчивается, система обнуляет текущие данные, предварительно отправив информацию по каналу USART на телефон. Схема алгоритмов обработки прерываний представлена на рисунке 19.

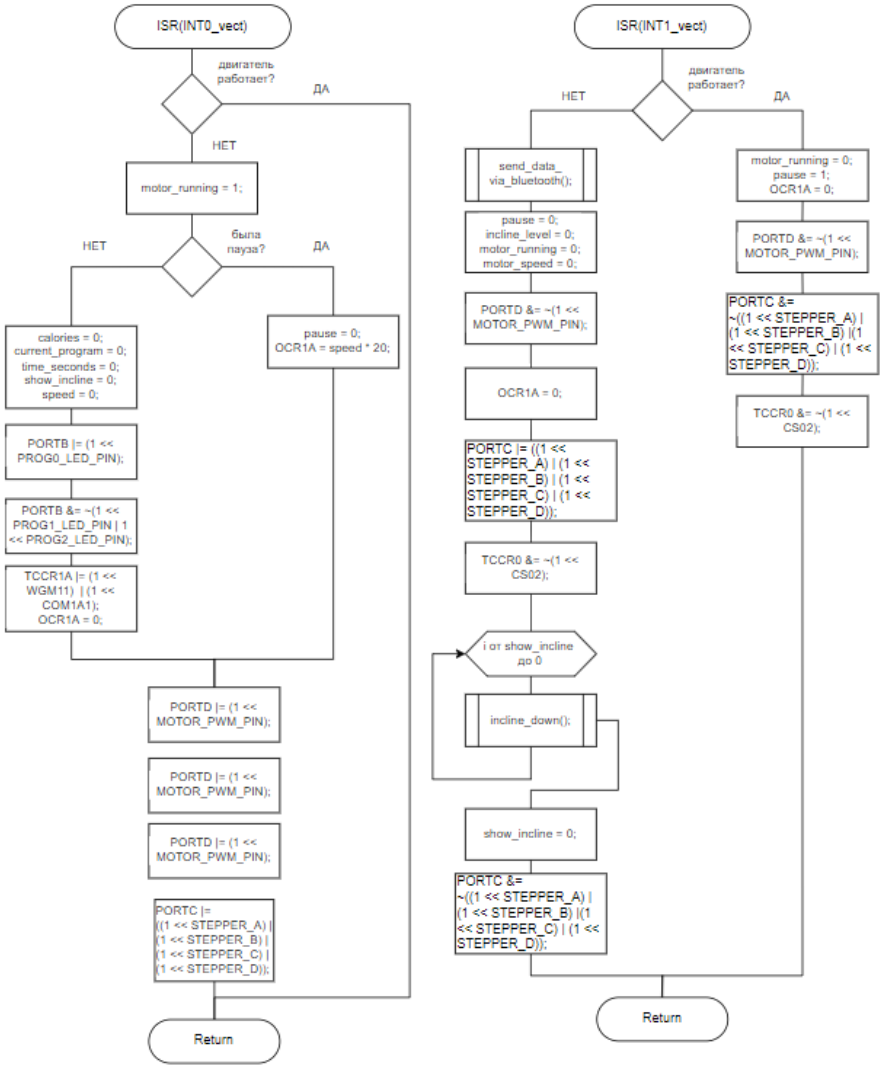

Рисунок 19 – Схемы алгоритмов кнопок «старт» и «стоп»

На основе составленных схем алгоритмов был написан код, который показан в Приложении А (Текст программы).

2 Технологическая часть

Для реализации работы модели микроконтроллера беговой дорожки была написана программа на языке Си, после загруженная в МК. Симуляция проводилась в программе Proteus 8.

2.1 Отладка и тестирование программы

Программа была отлажена в среде разработки AVR Studio, макет модели был собран в приложении Proteus 8. Proteus 8 предназначен для выполнения различных видов моделирования аналоговых и цифровых устройств. В этом приложении была смоделирована работа электродвигателей разных типов, подключен LCD и размещены кнопки с панели управления. Также для индикации текущей программы были размещены светодиоды зеленого, красного и желтого цвета. Дополнительно был добавлен модуль беспроводной передачи данных по каналу USART и подключен виртуальный терминал для проверки корректного отображения данных.

Все эти средства помогают убедиться в правильности написанной программы и управления реальными физическими устройствами.

При разработке алгоритма и написании кода программы были использованы следующие библиотеки:

- avr/io.h – библиотека ввода/вывода, которая используется для распознавания портов используемого микроконтроллера компилятором, сообщения их обозначения и их дополнительного функционала;
- util/delay.h – библиотека, позволяющая вызывать задержки (delay). При вызове _delay_ms или _delay_us в качестве параметра передается время в миллисекундах. Используется в программе для корректной настройки LCD, корректного управления шаговым двигателем и обработки прерываний, вызываемых с помощью кнопок;
- stdio.h – стандартная библиотека языка программирования C, которая предоставляет функции для ввода и вывода данных. Название stdio происходит от "Standard Input Output"; используется в

программе для форматного вывода данных на ЖК-дисплей и передачи данных в строковом формате в буфер USART;

- avr.interrupt.h – заголовочный файл в библиотеке AVR для программирования микроконтроллеров на основе архитектуры AVR, таких как ATmega и ATtiny. Этот файл предоставляет функции и макросы для работы с прерываниями, которые позволяют выполнять определенные действия в ответ на внешние или внутренние события, не блокируя основную программу.

Так как в работе активно используется система прерываний, можно также заметить системную функцию ISR, которая обозначает прерывание таймера. Она вызывается с разными параметрами. Поясним каждый из них:

INT0_vect – вектор внешнего прерывания по нажатию кнопки «старт» (изменение сигнала на входе INT0);

INT1_vect – вектор внешнего прерывания по нажатию кнопки «стоп» (изменение сигнала на входе INT1);

TIMER0_COMP_vect – вектор прерывания, когда значение счетчика таймера достигает значения, записанного в регистре сравнения;

TIMER2_OVF_vect – вектор прерывания таймера T2 в результате его переполнения [13].

В результате компиляции программы был создан файл с расширением ".hex" объемом 24 килобайта. Далее была проведена отладка разработанной программы в среде AVR Studio, проведено тестирование и симуляция работы модели беговой дорожки на виртуальном макете в программе Proteus 8 версии. Модель работает корректно и отвечает всем требованиям, заявленным в ТЗ.

2.2 Настройка таймеров и ШИМ сигнала двигателя

Проводим настройки таймеров, как это описано в *п. 1.3.5.*

Таймер T0 настраиваем на режим сравнения, загружаем в регистр сравнения OCR0 значение 255, разрешаем прерывания, установив единицу в регистр OCIE0. Предделитель, равный 256, будет установлен при нажатии на кнопку «старт», и тогда таймер начнет свою работу.

Таймер T1 предназначен для сигнала ШИМ. Выбираем режим FAST PWM (1 в биты WGM13, WGM12 и WGM11), TOP-значение в ICR1 равно 255, регистр сравнения OCR1A изначально пуст, так как беговая дорожка остановлена и скорость равна нулю. Его значение будет меняться в ходе тренировки.

Таймер T2 настроен на опрос кнопок, поэтому начинает свою работу с самого старта работы микроконтроллера. Загружаем предделитель равный 64 установкой в 1 битов CS22 и CS21. В TCNT2 загружаем 2 для точного отсчета 500 мс. Разрешаем прерывания – записываем 1 в бит TOIE2 регистра TIMSK.

Непосредственно в самой программе объявляем переменные time_seconds – для подсчета времени в секундах – и overflow_count – для отсчета необходимого количества переполнений.

2.3 Симуляция работы системы

Программа Proteus позволяет собрать модель системы и запустить ее в окружении, приближенном к реальным рабочим условиям. Схема проекта модели микроконтроллера беговой дорожки в Proteus показана на рисунке 20.

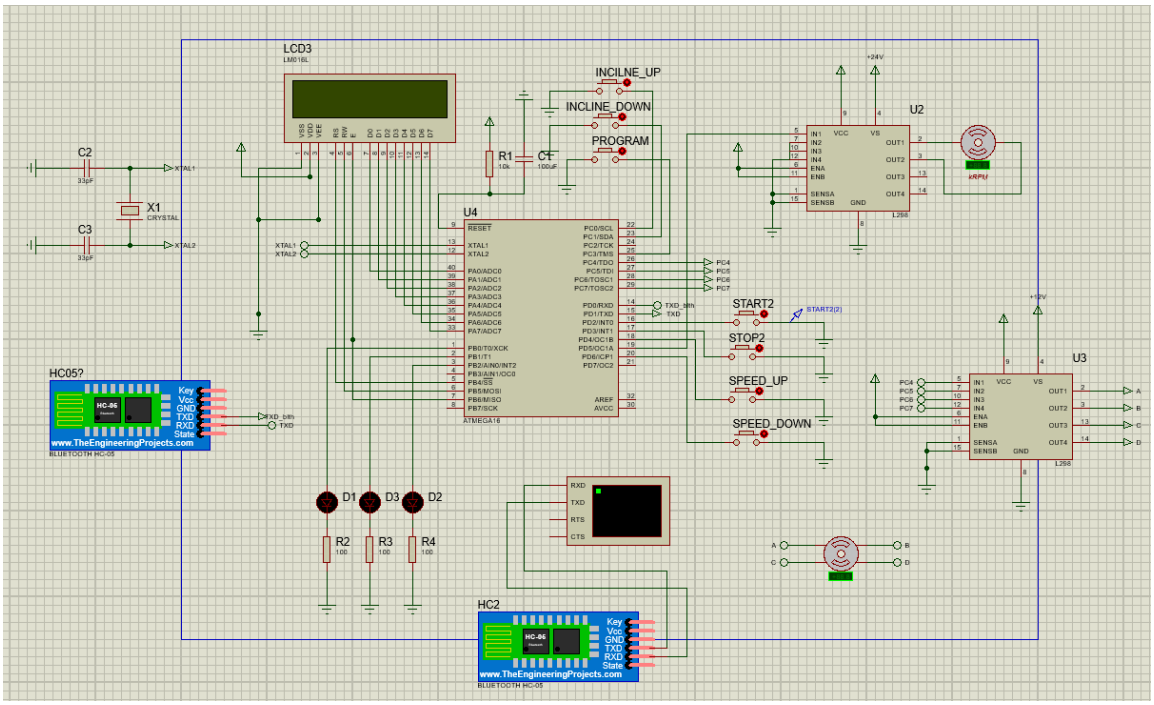

Рисунок 20 – Модель беговой дорожки

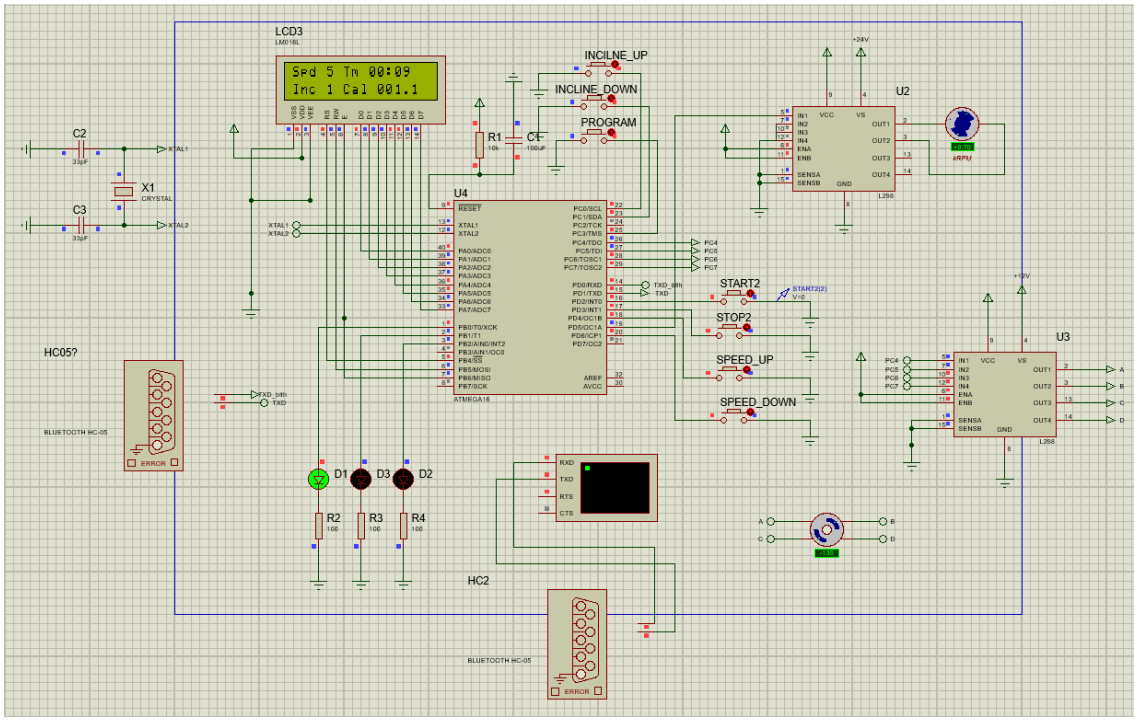

Рисунок 21 – Работа схемы

Для моделирования ввода данных с ПЭВМ используется инструмент системы – Virtual Terminal. С его помощью можно эмулировать терминал для взаимодействия с МК через интерфейс USART по портам TXD и RXD.

Запустив симуляцию и выбрав в верхнем меню Debug – Virtual Terminal, откроем виртуальный терминал. Для проверки работы канала передачи данных USART достаточно ввести и вернуть какое-либо значение. В данном случае программа в микроконтроллере будет считывать нажимаемые клавиши, получать символ нажатой клавиши по USART и возвращать этот символ. Каждый символ в терминале будет напечатан 2 раза подряд (один введенный, другой полученный). Результат представлен на рисунке 22.

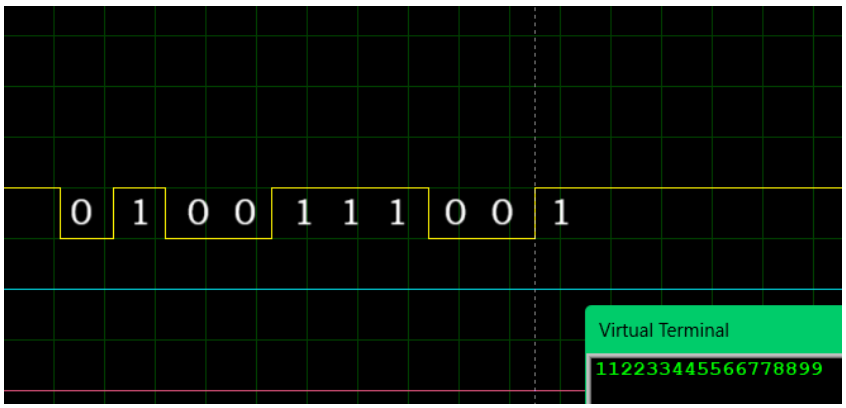

Рисунок 22 – Проверка работы канала USART

Желтая диаграмма – сигнал, принятый осциллографом с выхода TXD микроконтроллера при вводе символа «9» в виртуальный терминал. Проанализируем его.

Ожидаем 10-битную посылку. Первый бит, он же стартовый, равен нулю. Последний бит, он же стоповый, равен 1. Данные содержат число в двоичном коде, записанное в обратном порядке. То есть $10011100 = 00111001_2$. Переведем его в десятичную систему счисления.

$$10011100_2 = 57_{10}$$

Код символа «9» в таблице кодировки ASCII и есть 57. Передача работает корректно.

2.4 Способы программирования МК

После реализации алгоритма программы в виде кода и его отладки получаем файл с расширением .hex. Это бинарный файл с программой, который можно загрузить в микроконтроллер несколькими способами и использовать [16]:

- внутрисхемное программирование (ISP);
- параллельное высоковольтное программирование;
- через JTAG;
- через Bootloader;
- Pinboard II.

Выполним прошивку через канал SPI. Для записи программы в память микроконтроллера требуется программатор и одноименный разъем. Схема подключения представлена на рисунке 22.

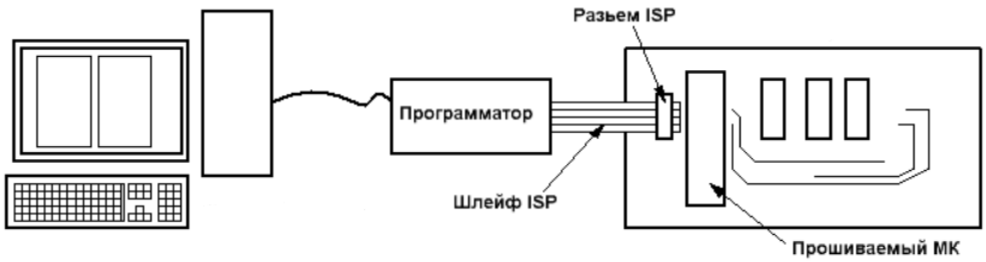

Рисунок 22 – Программирование МК

Прошивка проходит по интерфейсу SPI, для работы программатора нужно 4 контакта и питание (достаточно только земли, чтобы уравнять потенциалы земель программатора и устройства):

- MISO – Master-Input/Slave-Output – данные, идущие от контроллера;
- MOSI – Master-Output/Slave-Input – данные идущие в контроллер;
- SCK – тактовые импульсы интерфейса SPI;
- RESET – сигналом на RESET программатор вводит контроллер в режим программирования;
- GND – земля;
- VCC – питание.

Взаимодействие устройств по интерфейсу SPI требует установки одного из устройств в режим ведущего, а остальных – в режим ведомого. При этом ведущее устройство отвечает за выбор ведомого и инициализацию передачи.

Передача по SPI осуществляется в полнодуплексном режиме (в обе стороны), по одному биту за такт в каждую сторону. По возрастающему фронту сигнала SCK ведомое устройство считывает очередной бит с линии MOSI, а по спадающему – выдает следующий бит на линию MISO.

Таким образом в МК передается бинарный файл скомпилированной ранее программы.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана модель микроконтроллера беговой дорожки, основу которой составляет МК ATmega16 из серии ATmega семейства AVR. Устройство разработано в соответствии с ТЗ.

В процессе работы над курсовой работой были разработаны электрическая, функциональная и принципиальная схемы, спецификация с перечнем элементов и документация к устройству. Исходный код программы был разработан в среде AVR Studio на языке С, отлажен и протестирован при помощи симулятора Proteus 8.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хартов В.Я. Микроконтроллеры AVR. Практикум для начинающих: учебное пособие. – 2-е изд., испр. и доп. – М.: Издательство: МГТУ им. Н.Э. Баумана, 2012. – 280с.

2. Микроконтроллеры фирмы "ATMEL" семейства AVR [Электронный ресурс] – URL: https://www.promelec.ru/articles/73/ (дата обращения: 10.10.2024)

3. Документация к МК ATmega16 [Электронный ресурс] – URL: http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/avr/atmega16.htm (дата обращения: 07.11.2024)

4. Даташит ATmega16 [Электронный ресурс] – URL: https://www.alldatasheet.com/datasheet-pdf/view/78532/ATMEL/ATMEGA16.html (дата обращения: 03.11.2024)

5. MAX232 даташит [Электронный ресурс] – URL: https://www.alldatasheet.com/datasheet-pdf/pdf/73047/MAXIM/MAX232.html (дата обращения: 15.11.2024)

6. Использование интерфейса USART микроконтроллеров AVR - Микроконтроллеры и Технологии [Электронный ресурс] – URL: https://radioparty.ru/prog-avr/program-c/307-lesson-usart-avr (дата обращения: 13.11.2024)

7. ГОСТ 2.743-91 ЕСКД ОБОЗНАЧЕНИЯ БУКВЕННО-ЦИФРОВЫЕ В ЭЛЕКТРИЧЕСКИХ СХЕМАХ [Электронный ресурс]. – Режим доступа: https://docs.cntd.ru/document/1200001985 (дата обращения: 11.11.2024)

8. ГОСТ 2.721-74 ЕСКД ОБОЗНАЧЕНИЯ УСЛОВНЫЕ ГРАФИЧЕСКИЕ В СХЕМАХ [Электронный ресурс]. – Режим доступа: https://docs.cntd.ru/document/1200007058 (дата обращения: 11.11.2024)

9. В чем разница между двигателями постоянного и переменного тока? [Электронный ресурс] – URL: https://www.electricaltechnology.org/2020/06/difference-between-ac-dc-motor.html (дата обращения: 18.10.2024)

10. Энциклопедия бега [Электронный ресурс] – URL: https://get.run/wiki/ (дата обращения: 13.10.2024)

11. Как работает шаговый двигатель [Электронный ресурс] – URL: https://3drob.ru/stati/pro_3d_pechat/elektronika_3d_printera/kak_rabotaet_shagovyy_dvigatel (дата обращения: 25.10.2024)

12. Драйвер двигателя L298N [Электронный ресурс] – URL: https://3d-diy.ru/blog/drayver-dvigatelya-l298n/ (дата обращения: 25.10.2024)

13. AVR. Учебный курс. Таймеры. [Электронный ресурс] – URL: https://easyelectronics.ru/avr-uchebnyj-kurs-tajmery.html?ysclid=m4658l907r176153014 (дата обращения: 29.10.2024)

14. Неблокирующая обработка кнопок [Электронный ресурс] – URL: https://habr.com/ru/companies/timeweb/articles/703506/ (дата обращения: 05.11.2024)

15. LM016L Datasheet [Электронный ресурс] – URL: https://www.alldatasheet.com/datasheet-pdf/pdf/146552/HITACHI/LM016L.html (дата обращения: 06.11.2024)

16. Трактат о программаторах [Электронный ресурс] – URL: https://easyelectronics.ru/avr-uchebnyj-kurs-traktat-o-programmatorax.html (дата обращения: 11.11.2024)

17. Как правильно рассчитать калории на беговой дорожке [Электронный ресурс] – URL: https://www.livestrong.com/article/34973-calculate-treadmill-calories/ (дата обращения: 29.10.2024)

Приложение А

Текст программы

Объем исполняемого кода – 514 строк.

```
/*
 * treadmill_atmega16.c
 *
 * Created: 09.11.2024 15:46:34
 * Author : Evseenkov
 */
#include <avr/io.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define DELAY_TIME 35

#define F_CPU 4000000UL  // Частота 4 МГц

// Определение кнопок
#define START_BUTTON_PIN PD2
#define STOP_BUTTON_PIN PD3
#define SPEED_UP_BUTTON_PIN PD4
#define SPEED_DOWN_BUTTON_PIN PD6
#define INCLINE_UP_BUTTON_PIN PC0
#define INCLINE_DOWN_BUTTON_PIN PC1
#define PROGRAM_BUTTON_PIN PC3

// Макросы для светодиодов
#define PROG0_LED_PIN PB0
#define PROG1_LED_PIN PB1
#define PROG2_LED_PIN PB2

// Макрос для управления ШИМ
#define MOTOR_PWM_PIN PD5

// Макросы для выходов шагового двигателя
#define STEPPER_A PC4
#define STEPPER_B PC5
#define STEPPER_C PC6
#define STEPPER_D PC7

// Переменные для данных
volatile uint8_t motor_speed = 0;
volatile uint8_t incline_level = 0;
volatile uint8_t motor_running = 0;
volatile uint16_t time_seconds = 0;
volatile double calories = 0;
volatile uint16_t overflow_count = 0;
volatile char buff_out[10];
volatile uint8_t pause = 0;
```

```
volatile uint8_t inc_prev = 0;

//Переменные для отображения
volatile uint8_t current_program = 0;
volatile uint8_t speed = 0;
volatile uint8_t show_incline = 0;

// средний вес пользователя
#define USER_WEIGHT 75
// Макисмальная скорость беговой дорожки
#define MAX_SPEED 20

// Функция для отправки данных на LCD
void LCD_Data(uint8_t data) {
     PORTB |= (1 << PB4) | (1 << PB6);     // RS = 1 (режим
данных)
     PORTA = data;           // Устанавливаем данные на вывод
     _delay_us(1);
     PORTB &= ~(1 << PB6); // E = 1
     _delay_ms(2);           // Задержка для выполнения команды
}
// Функция для отправки данных на LCD
void LCD_Command(uint8_t cmd) {
     PORTB &= ~(1 << PB4); // RS = 0
     PORTB &= ~(1 << PB5); // RW = 0
     PORTB |= (1 << PB6); // E = 1
     PORTA = cmd; // Данные на вывод
     _delay_us(1);
     PORTB &= ~(1 << PB6); // E = 0 (отправка команды)
     _delay_ms(2);       // Задержка для выполнения команды
}
// Функция для печати строки на LCD
void LCD_Print(const char *str) {
     while (*str) {
          LCD_Data(*str++); // Отправка каждого символа на LCD
     }
}

// Инициализация портов для LCD
void portLCD_init() {
     // инициализация порта C
     DDRA |= 0xFF; // Устанавливаем PORTA данных как выход
     PORTA = 0x00; // Сбрасываем порты A
     // инициализация порта B
     DDRB |= (1 << PB4) | (1 << PB5) | (1 << PB6); //
управляющие сигналы как выходы
     PORTB = 0x00;
     _delay_ms(20);       // Задержка после подачи питания
}
// Инициализация LCD
void LCD_Init(){
     portLCD_init(); // Настройка портов вывода
```

```
    // Инициализация в 8-битном режиме
    LCD_Command(0x08); // Предварительно выключить дисплей
    LCD_Command(0x38); // Установка режима 8 бит
    _delay_us(1);
    LCD_Command(0x38);
    _delay_us(1);
    LCD_Command(0x38);
    LCD_Command(0x01); // Очистка экрана
    _delay_us(1);
    LCD_Command(0x06); // Инкремент адреса
    _delay_ms(10);
    LCD_Command(0x0C); // Включение дисплея и отключение
курсора
    _delay_ms(2);      // Задержка для выполнения очистки
}
// Функция обновления данных на дисплее
void update_lcd() {
    LCD_Command(0x01); // Очистка дисплея
    _delay_ms(2);
    LCD_Command(0x80); // Курсор на первую строку

    // Получаем целую и дробную части параметров для их вывода
на LCD
    //double grade = incline_level/255 * 100; // Наклон дорожки
в процентном выражении от максимума
    //uint8_t incline_integer = (uint8_t)(grade); // Целая
часть
    //uint8_t incline_fraction = (uint8_t)(grade -
incline_integer); // Дробная часть
    uint8_t cal_int = (uint8_t)(calories);
    uint8_t cal_frac = (uint8_t)((calories - cal_int)*10);
    uint8_t time_min = (uint8_t)time_seconds/60;
    uint8_t time_sec = (uint8_t)time_seconds%60;
    char buffer[16];
    snprintf(buffer, sizeof(buffer), "Spd %u Tm %02d:%02d",
speed, time_min, time_sec);
    LCD_Print(buffer);
    _delay_ms(2);
    LCD_Command(0xC0); //установка курсора на 2 строку
    // Отображение калорий
    snprintf(buffer, sizeof(buffer), "Inc %u Cal %03u.%1u",
show_incline, cal_int, cal_frac);
    LCD_Print(buffer);
}

// Настройка UART для передачи данных
void USART_init() {
    uint16_t ubrr = F_CPU/16/9600 - 1;
    UBRRH = (unsigned char)(ubrr >> 8);  // Старшие биты
скорости
    UBRRL = (unsigned char)ubrr;         // Младшие биты
скорости
```

```
    UCSRA = 0;
    UCSRB |= (1 << TXEN | 1 << RXEN);  // Включаем передачу
    UCSRC |= (1<<URSEL) | (1 << UCSZ1) | (1 << UCSZ0);  // 8-
битные данные, 1 стоп-бит
}

// Функция для отправки одного байта
void USART_send_char(char data) {
    while (!(UCSRA & (1 << UDRE)));  // Ждем, пока буфер
передачи не освободится
    UDR = data;  // Отправляем символ
}

// Функция для отправки строки
void USART_send_string(const char* str) {
    while (*str) {
        USART_send_char(*str++);
    }
}

void send_data_via_bluetooth() {
    USART_send_string("Training:");
    char buffer[20];

    uint8_t time_min = (uint8_t)time_seconds/60;
    uint8_t time_sec = (uint8_t)time_seconds%60;
    snprintf(buffer, sizeof(buffer), "Time %02u:%02u",
time_min, time_sec);
    USART_send_string(buffer);

    uint8_t cal_int = (uint8_t)(calories);
    uint8_t cal_frac = (uint8_t)((calories - cal_int)*10);
    USART_send_string("Calories:");
    snprintf(buffer, sizeof(buffer), "%u.%u", cal_int,
cal_frac);
    USART_send_string(buffer);

}

// Настройка кнопок с прерываниями
void buttons_init() {
    // Устанавливаем пины кнопок как входы
    DDRD &= ~((1 << START_BUTTON_PIN) | (1 << STOP_BUTTON_PIN)
| (1 << SPEED_UP_BUTTON_PIN) | (1 << SPEED_DOWN_BUTTON_PIN));
    DDRC &= ~((1 << INCLINE_UP_BUTTON_PIN) | (1 <<
INCLINE_DOWN_BUTTON_PIN) | (1 << PROGRAM_BUTTON_PIN));

    // Устанавливаем пины светодиодов как выходы
    DDRB |= (1 << PROG0_LED_PIN | 1 << PROG1_LED_PIN | 1 <<
PROG2_LED_PIN);
    // Настройка шагового двигателя для регулировки наклона
бегового полотна
```

```
    DDRC |= ((1 << STEPPER_A) | (1 << STEPPER_B) | (1 <<
STEPPER_C) | (1 << STEPPER_D)); // пины на вывод

    // Включаем pull-up резисторы
    PORTD |= ((1 << START_BUTTON_PIN) | (1 << STOP_BUTTON_PIN)
| (1 << SPEED_UP_BUTTON_PIN) | (1 << SPEED_DOWN_BUTTON_PIN));
    PORTC |= ((1 << INCLINE_UP_BUTTON_PIN) | (1 <<
INCLINE_DOWN_BUTTON_PIN) | (1 << PROGRAM_BUTTON_PIN));

    // Настройка внешних прерываний
    MCUCR |= (1 << ISC01); //| (1 << ISC11);  // Прерывание по
спаду на INT0 и INT1
    GICR |= (1 << INT0) | (1 << INT1);  // Включаем INT0 и INT1
}
void btn_timer_init(){
    // Настройка таймера 2
   TCCR2 = (1 << CS22) | (1 << CS21); // Установить
предделитель 64
    TCNT2 = 2; // Начальное значение (6)
    TIMSK |= (1 << TOIE2); // Разрешить прерывание по
переполнению таймера 2
}

// Инициализация таймера для отсчета времени тренировки
void timer_init() {
    // Настройка таймера T0 для отсчета секунд

    TCCR0 |= (1 << WGM01);  // CTC mode
     // Prescaler = 256
     //TCCR0 |= (1 << CS01) | (1 << CS00);  // Prescaler 1024
    TCCR0 &= ~((1 << CS02) | (1 << CS01) | (1 << CS00));
    OCR0 = 255;  // TOP = 255
    TIMSK |= (1 << OCIE0);  // Включаем прерывание
}

// Инициализация ШИМ для двигателя
void pwm_init() {
    // порт ШИМ двигателя на вывод
    DDRD |= (1 << MOTOR_PWM_PIN);

    // Настройка Fast PWM для двигателя (Таймер T1)
    TCCR1A |= (1 << WGM11)  | (1 << COM1A1);
    TCCR1B |= ((1 << WGM12) | (1 << WGM13)); //| (1 << CS10)
    ICR1 = 255;  // TOP для 8-битного ШИМ
    OCR1A = 0;  // Изначально мотор остановлен
    TCCR1B &= ~(1 << CS10);

    TCCR1A = 0; // Отключить ШИМ
    PORTD &= ~(1 << PD5); // Установить PD5 в низкий уровень
}
//функция подъема бегового полотна
void incline_up()
{
```

```
     PORTC |= ((1 << STEPPER_C));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_B) | (1 <<
STEPPER_D));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_B) | (1 << STEPPER_C));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_D));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_B));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_C) | (1 <<
STEPPER_D));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_A) | (1 << STEPPER_B));
     PORTC &= ~((1 << STEPPER_C) | (1 << STEPPER_D));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_A));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_C) | (1 <<
STEPPER_D));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_A) | (1 << STEPPER_D));
     PORTC &= ~((1 << STEPPER_B) | (1 << STEPPER_C));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_D));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_B) | (1 <<
STEPPER_C));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_C) | (1 << STEPPER_D));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_B));
     _delay_ms(50);
}
//функция опускания бегового полотна
void incline_down()
{
     PORTC |= ((1 << STEPPER_C) | (1 << STEPPER_D));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_B));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_D));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_B) | (1 <<
STEPPER_C));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_A) | (1 << STEPPER_D));
     PORTC &= ~((1 << STEPPER_B) | (1 << STEPPER_C));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_A));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_C) | (1 <<
STEPPER_D));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_A) | (1 << STEPPER_B));
     PORTC &= ~((1 << STEPPER_C) | (1 << STEPPER_D));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_B));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_C) | (1 <<
STEPPER_D));
     _delay_ms(50);
```

```
     PORTC |= ((1 << STEPPER_B) | (1 << STEPPER_C));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_D));
     _delay_ms(50);
     PORTC |= ((1 << STEPPER_C));
     PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_B) | (1 <<
STEPPER_D));
     _delay_ms(50);
}
// Прерывание на старт
ISR(INT0_vect) {
     if (!motor_running)
     {
          motor_running = 1;
          if (pause)
          {
               pause = 0;
               OCR1A = speed * 20;
          }
          else
          {
               calories = 0;
               current_program = 0;
               time_seconds = 0;
               show_incline = 0;
               speed = 0;
               // Подключаем свтеодиоды для отображения
программы
               PORTB |= (1 << PROG0_LED_PIN);
               PORTB &= ~(1 << PROG1_LED_PIN | 1 <<
PROG2_LED_PIN);
               TCCR1A |= (1 << WGM11)  | (1 << COM1A1);
               OCR1A = 0;  // Начальная скорость = 0
          }
          PORTD |= (1 << MOTOR_PWM_PIN); // Подтягивающий
реистор
          TCCR1B |= (1 << CS10); // Включаем двигатель
          TCCR0 |= (1 << CS02); //Включить таймер времени
          //TCCR0 &= ~(1 << CS02); //приостановить таймер
времени

          //Подтягивающие резисторы для шагового двигателя
          PORTC |= ((1 << STEPPER_A) | (1 << STEPPER_B) | (1 <<
STEPPER_C) | (1 << STEPPER_D));
     }
}
// Прерывание на стоп
ISR(INT1_vect) {
     if (motor_running){ //тренировка на паузе
          motor_running = 0;
          pause = 1;
          OCR1A = 0; // Сбрасываем скорость
          PORTD &= ~(1 << MOTOR_PWM_PIN); // Ставим двигатель на
паузу
```

```
          PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_B) |
//Отключить управление наклоном на паузе
               (1 << STEPPER_C) | (1 << STEPPER_D));


     }
     else //Завершить тренировку и отправить данные
     {
          send_data_via_bluetooth(); // отправить данные
          pause = 0;
          incline_level = 0;
          motor_running = 0;
          PORTD &= ~(1 << MOTOR_PWM_PIN); // Выключаем двигатель
          OCR1A = 0; // Сбрасываем скорость
          motor_speed = 0;
          // Выключаем наклон
          //Подтягивающие резисторы для шагового двигателя
          PORTC |= ((1 << STEPPER_A) | (1 << STEPPER_B) | (1 <<
STEPPER_C) | (1 << STEPPER_D));
          for (int i = show_incline; i > 0; i--)
          {
               incline_down();
          }
          show_incline = 0;
          PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_B) |
               (1 << STEPPER_C) | (1 << STEPPER_D));
     }
     TCCR0 &= ~(1 << CS02); //приостановить таймер времени
     //TCCR0 &= ~(1 << CS01) | (1 << CS00);  // Prescaler 1024
}
// Прерывание для отсчета времени тренировки
ISR(TIMER0_COMP_vect) {

     overflow_count++;
     if (overflow_count == 61 && motor_running) {
          overflow_count = 0;
          time_seconds++;

          double calories_per_min = 0;
          double grade = show_incline / 13.0; // наклон в
процентах
          double speed_mtrs = speed * 20 / 255.0 * MAX_SPEED *
1000 / 60.0; // скорость в метрах/минуту
          if (speed > 0)
          {
               if (speed < 4)
               {
                   calories_per_min = (0.1 * speed_mtrs) + (1.8
* speed_mtrs * grade) + 3.5;
               }
               else
               {
```

```
                    calories_per_min = (0.2 * speed_mtrs) + (0.9
* speed_mtrs * grade) + 3.5;
                }
                calories += (calories_per_min / 200.0 *
USER_WEIGHT / 60.0); // добавляем сожженные калории в секунду
          }
          update_lcd(); // Обновление данных на дисплее
     }
}

// Прерывание для обработки кнопок
ISR(TIMER2_OVF_vect) {
     if (motor_running){
          // Увеличение скорости
          if (!(PIND & (1 << SPEED_UP_BUTTON_PIN)) &&
motor_speed <= 230) {
                motor_speed += 20;
                speed++;
                OCR1A = motor_speed;
                _delay_ms(DELAY_TIME);
          }
          // Уменьшение скорости
          if (!(PIND & (1 << SPEED_DOWN_BUTTON_PIN)) &&
motor_speed >= 20) {
                motor_speed -= 20;
                speed--;
                OCR1A = motor_speed;
                _delay_ms(DELAY_TIME);
          }
          // Увеличение наклона
          if (!(PINC & (1 << INCLINE_UP_BUTTON_PIN))  &&
incline_level <= 230) {
                show_incline++;
                incline_level += 20;
                //Подъем полотна (шаговый двигатель вперед)
                incline_up();
          }
          // Уменьшение наклона
          if (!(PINC & (1 << INCLINE_DOWN_BUTTON_PIN)) &&
incline_level >= 20) {
                show_incline--;
                incline_level -= 20;
                //Спуск полотна (шаговый двигатель назад)
                incline_down();
                PORTC &= ~((1 << STEPPER_A) | (1 << STEPPER_B) |
(1 << STEPPER_C) | (1 << STEPPER_D));
          }
          // Переключение программы
          if (!(PINC & (1 << PROGRAM_BUTTON_PIN))) {
                current_program = (current_program + 1) % 3; //
Всего 3 программы (0, 1 и 2)
                inc_prev = show_incline; //запомниаем текущий
уровень наклона
```

```
                if (current_program == 1) //режим ходьбы
                {
                     PORTB |= (1 << PROG1_LED_PIN);
                     PORTB &= ~(1 << PROG0_LED_PIN | 1 <<
PROG2_LED_PIN);
                     //настройка наклона
                     incline_level = 80;
                     show_incline = 8;
                     //определяем, как нужно имзенить наклон
(увеличить/уменьшить)
                     if (inc_prev < show_incline)
                     {
                          for (int i = 0; i < show_incline; i++)
                          {
                               incline_up();
                          }
                     }
                     else
                     {
                          for (int i = 0; i < show_incline; i++)
                          {
                               incline_down();
                          }
                     }
                     //настройка скорости
                     speed = 4;
                     motor_speed = 40;
                     OCR1A = motor_speed;
                }
                else if (current_program == 2) //режим легкого
бега
                {
                     PORTB |= (1 << PROG2_LED_PIN);
                     PORTB &= ~(1 << PROG0_LED_PIN | 1 <<
PROG1_LED_PIN);
                     //настройка наклона
                     incline_level = 40; //меняем отображение
наклона и рассчеты
                     show_incline = 4;
                     if (inc_prev < show_incline)
                     {
                          for (int i = 0; i < show_incline; i++)
                          {
                               incline_up();
                          }
                     }
                     else
                     {
                          for (int i = 0; i < show_incline; i++)
                          {
                               incline_down();
                          }
                     }
```

```
                    //настройка скорости
                    speed = 8;
                    motor_speed = 80;
                    OCR1A = motor_speed;
               }
               else
               {
                    PORTB |= (1 << PROG0_LED_PIN);
                    PORTB &= ~(1 << PROG1_LED_PIN | 1 <<
PROG2_LED_PIN);
               }

          }
     }
    TCNT2 = 2; // Сброс таймера опроса кнопок
     //_delay_ms(DELAY_TIME);
}
//  Получение байта
char USARTReceiveChar(void) {
  //  Устанавливается, когда регистр свободен
  while(!(UCSRA & (1<<RXC)));
  return UDR;
}

int main(void) {
     // Инициализация
     pwm_init(); // Двигатель
     buttons_init(); // Кнопки
     btn_timer_init(); // Опрос кнопок по таймеру
     timer_init(); // Таймер времени
     LCD_Init(); // Настройка дисплея
     LCD_Print("Press START"); // Стартовое сообщение
     USART_init();
     sei();  // Включаем глобальные прерывания

     // Начало работы
     while (1) {}
}
```

Приложение Б

Графическая часть

На 2 листах

Электрическая схема функциональная

Электрическая схема принципиальная

Приложение В

Перечень элементов

На 2 листах