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:

Тренажер для оператора

Студент ИУ6-72Б (Группа) _____ (Подпись, дата) С.В. Астахов (И.О. Фамилия)

Руководитель _____ (Подпись, дата) С.А. Хохлов (И.О. Фамилия)

2022 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ-6
_____А.В. Пролетарский
«_2__»_сентября_____2022 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Микропроцессорные системы

Студент Астахов С.В. (ИУ6-72Б)_____
(фамилия, инициалы, индекс группы)

Направленность курсовой работы – учебная

График выполнения работы: 25% - 4 нед., 50% - 8 нед., 75% - 12 нед., 100% -16 нед.

Тема курсовой работы Тренажер для оператора

Разработать устройство для измерения времени реакции оператора на движущуюся мишень (светящийся светодиод) по одной из 8-и линеек светодиодной матрицы 8x4. После появления мишени испытуемый должен “захватить цель”, нажав кнопку с номером активной линейки на матричной клавиатуре 4x4 пульта оператора. Время с момента появления мишени до ее захвата вывести на дисплей из 7-сегментных индикаторов. В случае промаха сформировать короткий звуковой сигнал для SPEAKER. Подсчитать среднее время реакции оператора за К испытаний и также вывести на дисплей.
Разработать алгоритм запуска мишеней, выбирая линейку траектории по случайному закону. Предусмотреть регулировку скорости перемещения мишени вдоль линейки. Результаты испытаний переслать в ПЭВМ по команде с пульта оператора.
Микроконтроллер - ATmega8535, K=15.
Отладить модули разработанной программы с помощью симулятора.
Оценить потребляемую мощность устройства.

Оформление курсовой работы
1. Расчетно-пояснительная записка на__30____ листах формата А4.
2. Перечень графического материала
а) схема функциональная электрическая
б) схема принципиальная электрическая

Дата выдачи задания 4 сентября 2022 г.
Руководитель курсовой работы _____/ Хартов В.Я./
Задание получил _____/ Астахов С.В. / «__4__»_сентября_____2022 г.

Дата защиты - 20 декабря 2022 г.
Примечание: Задание оформляется в двух экземплярах; один выдаётся студенту, второй хранится на кафедре.

РЕФЕРАТ

РПЗ X стр., 8 таблиц, 20 рисунков, 8 источников, 4 приложения.

МИКРОКОНТРОЛЛЕР, ATMEGA8535, ПУЛЬТ ОПЕРАТОРА, ТРЕНАЖЕР, СКОРОСТЬ РЕАКЦИИ.

Объектом разработки является тренажер оператора.

Тренажер оператора представляет собой устройство, измеряющее скорость реакции оператора на движущуюся мишень. Устройство может выводить информацию о результатах испытаний на дисплей из ССИ и пересылать эти данные на ПЭВМ.

Цель работы – создание программного обеспечения, функциональной и принципиальной схем, спецификации описанного устройства.

Задачи, решенные в процессе выполнения курсовой работы:

- анализ объекта разработки на функциональном уровне;
- разработка функциональной схемы;
- выбор элементной базы для реализации устройства;
- разработка принципиальной схемы;
- расчет потребляемой мощности;
- разработка алгоритмов работы программного обеспечения;
- реализация программного обеспечения устройства на языке С;
- отладка программного обеспечения в среде симуляции Proteus 8.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

АЦП — аналого-цифровой преобразователь.

ССИ — семисегментный индикатор.

МК — микроконтроллер.

ПЭВМ — персональная электронная вычислительная машина.

RISC — (reduced instruction set computer) архитектурный подход к проектированию процессоров, в которой быстродействие увеличивается за счёт такого кодирования инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим.

UART — (Universal Asynchronous Receiver-Transmitter) узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами. Преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по одной физической цифровой линии другому аналогичному устройству.

USB — (Universal Serial Bus) последовательный интерфейс для подключения периферийных устройств к вычислительной технике.

SRAM — (static random access memory) статическая память с произвольным доступом, полупроводниковая оперативая память, в которой каждый разряд хранится в схеме с положительной обратной связью, позволяющей поддерживать состояние без регенерации.

EEPROM — (Electrically Erasable Programmable Read-Only Memory) электрически стираемое перепрограммируемое постоянное запоминающее устройство.

FLASH — разновидность полупроводниковой технологии электрически перепрограммируемой памяти

Содержание

Введение 6
1 Конструкторская часть 7
1.1 Разработка функциональной схемы 7
1.1.1 Разработка обобщенной функциональной схемы 7
1.1.2 Описание архитектуры и технические характеристики микроконтроллера 8
1.1.3 Детализация функциональной схемы 10
1.2 Разработка принципиальной схемы 11
1.2.1 Драйвер дисплея (SN74LS48) 11
1.2.2 Расчет транзисторных ключей 12
1.2.3 Драйвер USB-UART (FT232RL) 13
1.3 Расчет потребляемой мощности 15
1.4 Разработка программной части 16
1.4.1 Использованные библиотеки и подпрограммы 16
1.4.2 Алгоритм основной программы 18
1.4.3 Управление динамической индикацией 22
1.4.4 Отсчет времени реакции 23
1.4.5 Работа с UART 25
2 Технологическая часть 30
2.1 Тестирование программы 30
2.2 Программирование микроконтроллера 32
ЗАКЛЮЧЕНИЕ 37
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 38
ПРИЛОЖЕНИЕ А. Исходный текст программы 39
ПРИЛОЖЕНИЕ Б. Функциональная электрическая схема 50
ПРИЛОЖЕНИЕ В. Принципиальная электрическая схема 51
ПРИЛОЖЕНИЕ Д. Перечень радиоэлементов 52

Введение

В данной работе производится разработка тренажера для оператора.

Основная функция устройства – измерение времени реакции оператора на одну движущуюся мишень и среднего времени реакции на K (K=15) мишеней. Мишени представлены горящими светодиодами, ввод информации происходит с матричной клавиатуры, вывод – на дисплей из 7-сегментных индикаторов. Кроме того, устройство должно издавать звук при неправильном «захвате мишени», иметь возможность изменения скорости движения мишеней и возможность передачи результатов испытаний в ПЭВМ.

В устройстве использован микроконтроллер ATmega8535 семейства AVR. Микроконтроллеры AVR имеют гарвардскую архитектуру (программа и данные находятся в разных адресных пространствах) и систему команд, близкую к идеологии RISC. Процессор AVR имеет 32 8-битных регистра общего назначения, объединённых в регистровый файл. В отличие от «идеального» RISC, регистры не абсолютно ортогональны.

Целевой МК имеет четыре 8-разрядных порта ввода-вывода, 10-разрядный АЦП, два 8-разрядных и один 16-разрядный таймер, EEPROM и RAM объемом по 512 байт, встроенные интерфейсы I2C, SPI, UART. МК способен работать на частоте до 16 МГц.

1 Конструкторская часть

1.1 Разработка функциональной схемы

1.1.1 Разработка обобщенной функциональной схемы

Проектирование устройства было начато с разработки функциональной схемы, в соответствии с методическими указаниями [1].

На основе текста задания можно определить, что система будет состоять из следующих элементом:

- дисплей из 7-сегментных индикаторов;
- матричная клавиатура;
- светодиодная матрица;
- зуммер (звукоизвлекатель).

Кроме того, необходимо осуществлять передачу данных к ПЭВМ, для чего понадобится драйвер, преобразующий сигналы интерфейса UART/I2C/SPI в сигналы интерфейса USB/RS-232.

Для управления дисплеем и светодиодной матрицей так же будем использовать вспомогательные драйверы, чтобы сократить количество задействованных контактов МК.

На основе результатов первичного анализа требований была составлена обобщенная функциональная схема, представленная на рисунке 1.

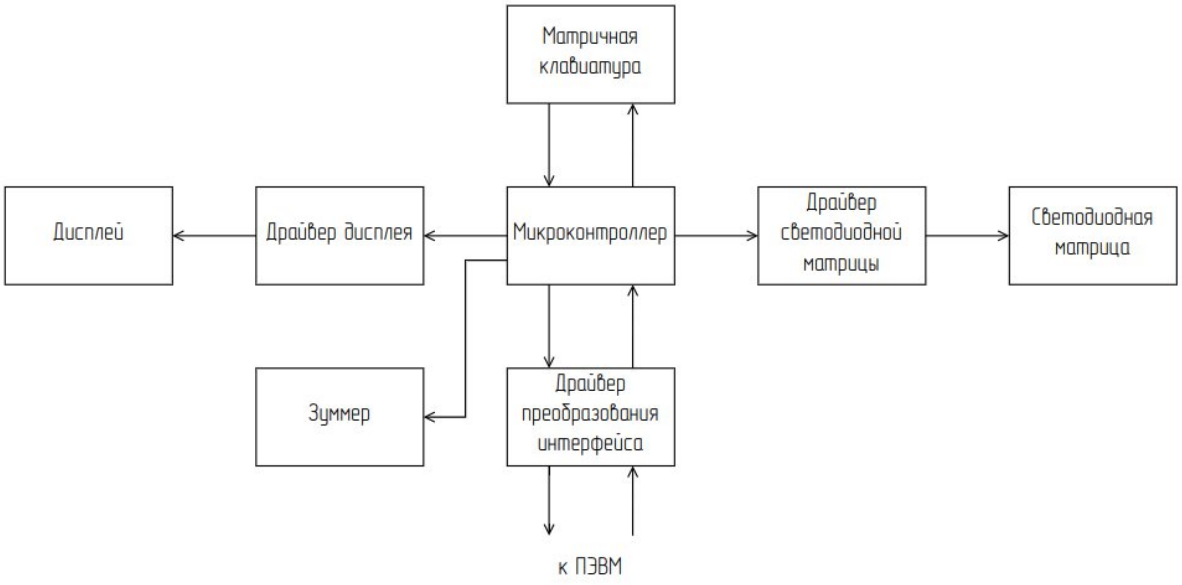

Рисунок 1 — обобщенная функциональная схема

1.1.2 Описание архитектуры и технические характеристики микроконтроллера

Семейство микроконтроллеров AVR включает tinyAVR, megaAVR и XMEGA AVR.

В контроллерах типа tinyAVR максимальное число линий ввода-вывода составляет 18, чего будет недостаточно для подключения 5 периферийных устройств. Возможности контроллеров типа XMEGA AVR наиболее широки, но, при использовании контроллеров данного типа большинство их возможностей не будет использовано, поэтому для реализации устройства было решено использовать контроллер типа megaAVR.

Среди микроконтроллеров типа megaAVR был выбран ATmega8535, который обладает следующими характеристиками [2]:

- 8-битная шина данных;
- тактовая частота до 16 МГц;
- четыре 8-разрядных порта ввода-вывода (32 контакта);
- 8 КБайт памяти программ;
- 10-разрядный АЦП;
- встроенные интерфейсы I2C, SPI, UART;
- два 8-разрядных и один 16-разрядный таймер/счетчик.

Так как в микроконтроллерах AVR используется гарвардская архитектура, в соответствии с которой разделены адресные пространства памяти программ и памяти данных. Такая структура позволяет процессору одновременно работать с памятью программ и памятью данных, что позволяет повысить производительность.

В памяти данных регистровая и оперативная память (SRAM) образуют единое адресное пространство, которое может быть расширено за счет подключения внешнего запоминающего устройства ERAM.

Для долговременного хранения данных, которые могут изменяться в процессе работы МК используется энергозависимая память EEPROM.

Карта памяти ATmega8535 представлена на рисунке 2.

Рисунок 2 — карта памяти ATmega8535

Структурная схема ATmega8535 представлена на рисунке 3.

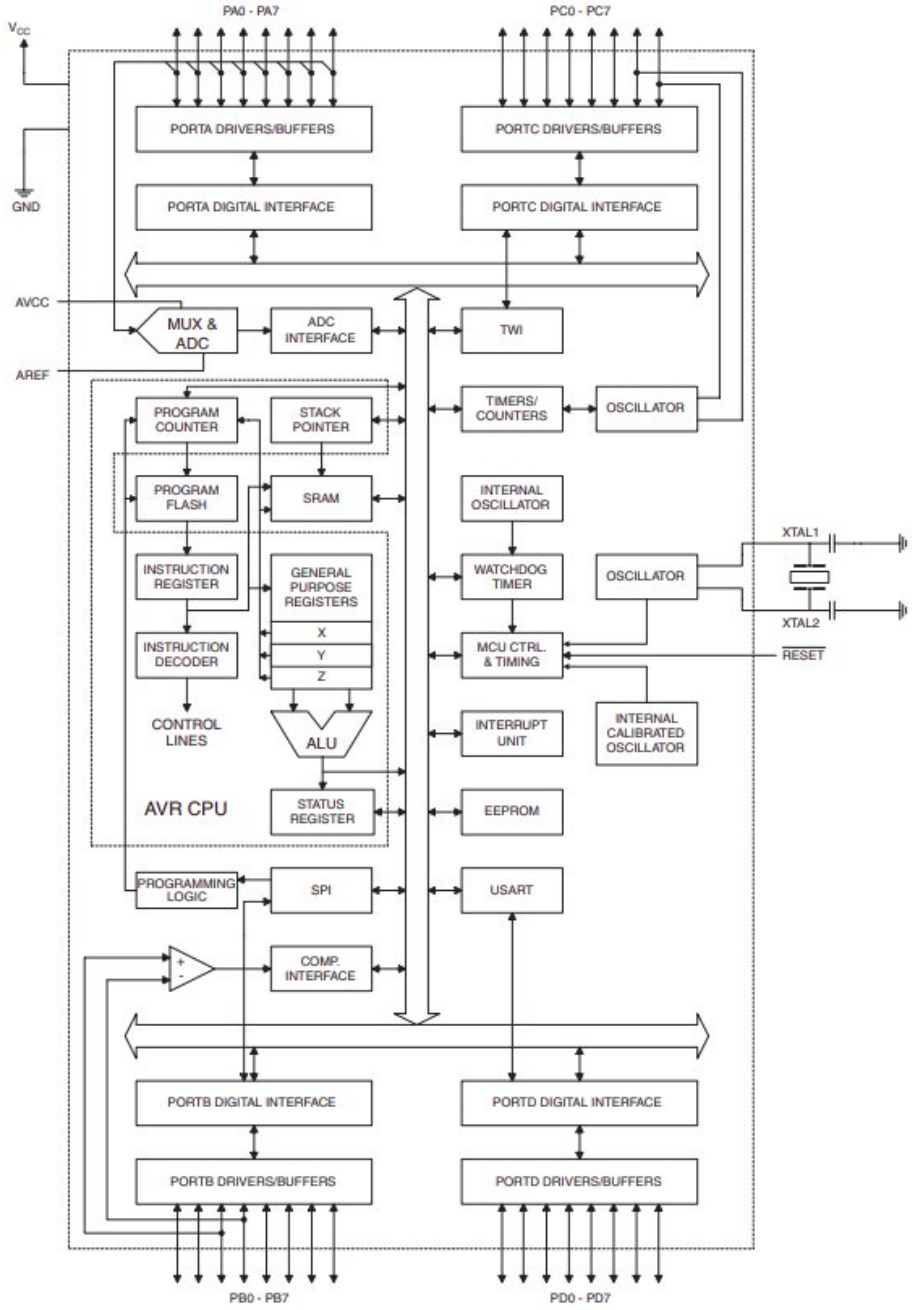

Рисунок 3 — структурная схема ATmega8535

1.1.3 Детализация функциональной схемы

Для детализации функциональной схемы были подобраны цифровые схемы, дисплей на основе 7-сегментных индикаторов и детализировано устройство светодиодной матрицы. Кроме того, описано к каким линиям портов МК подключаются те или иные устройства.

Примечание: в данном разделе используемые схемы описаны лишь поверхностно, подробнее см. раздел «Проектирование принципиальной схемы».

В качестве устройства ввода используется матричная клавиатура 4x4, активная строка в которой выбирается с помощью линий PA0-PA3, а столбцы считываются с помощью линий PA4-PA7 порта A.

В качестве дисплея используется схема FYQ-5641AUG-21 с общим катодом. Выбор активного индикатора при динамической индикации осуществляется с помощью линий PB4-PB7 порта B. Выбор сегментов осуществляется с помощью драйвера семейства 7448, подключенного к линиям PB0-PB3 порта B.

Передача данных на ПЭВМ осуществляется с помощью интерфейса UART через драйвер USB-UART, подключенный к линиям PD0(Rx), PD1(Tx) порта D.

Звуковой сигнал вырабатывается с помощью зуммера со встроенным генератором, подключаемого к линии PD7 порта D.

Светодиодная матрица реализована на базе светодиодных полос по 4 светодиода. Так как на основе требований задания можно заключить, что одновременно может быть активен всего один светодиод, матрица управляется с помощью дешифратора 5-32, реализованного на базе дешифратора 3-8 и восьми дешифраторов 2-4. Линии PC0-PC4 подключены к адресным разрядам дешифраторов, а линия PC7 — к разрешающему входу.

Полученная функциональная схема содержится в приложении Б.

1.2 Разработка принципиальной схемы

1.2.1 Драйвер дисплея (SN74LS48)

Для упрощения управления ССИ в составе дисплея, а также для сокращения числа использованных контактов микроконтроллера был использован драйвер SN74LS48, преобразующий 4 бита двоично-десятичного кода (один десятичный разряд) в сигналы для сегментов ССИ [3].

Условное графическое обозначение драйвера представлено на рисунке 4. (на изображении устройство отражено относительно вертикальной оси, как это сделано для удобства расположения на принципиальной схеме)

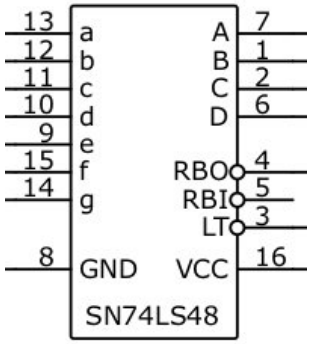

Рисунок 4 — условное графическое обозначение SN74LS48

Назначение контактов микросхемы:

Входы A, B, C, D — разряды двоично-десятичного числа.

Выходы a, b, ..., g — сигналы для сегментов ССИ.

Вход LT — тест светодиодов (активация всех светодиодов).

Вход RBI — не отображать 0.

Вход BI — не отображать 0.

Выход RBO — выключено отображение 0.

Состояние ССИ в зависимости от входного числа показано на рисунке 5.

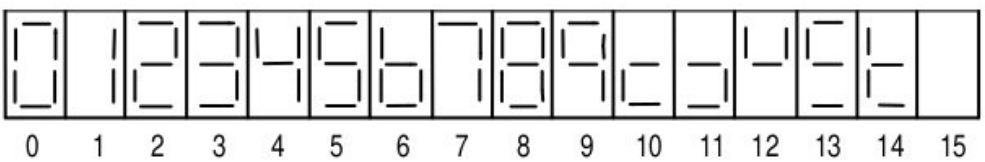

Рисунок 5 — состояния ССИ

1.2.2 Расчет транзисторных ключей

Так как дешифраторы, используемые для выбора активного светодиода в матрице имеют малые значения предельного выходного тока, для питания светодиодов необходимо использовать транзисторные ключи.

Схема такого с использованием n-p-n транзистора приведена на рисунке 6.

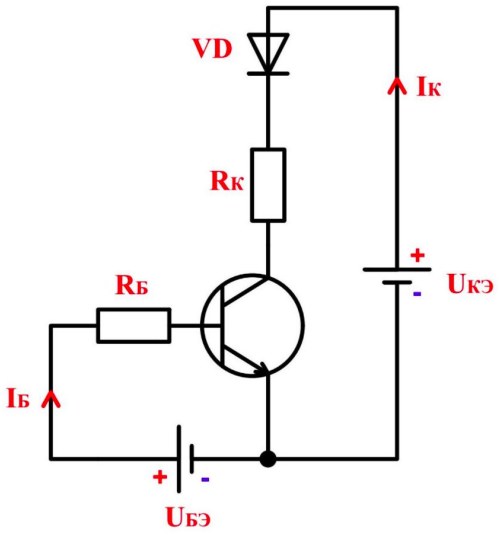

Рисунок 6 — транзисторный ключ

В основе транзисторного ключа используется транзистор 2N2222, который имеет коэффициент усиления по току β = 100..300, $\Delta U_{кэ}$ = 0.1 и $\Delta U_{бэ}$ = 0.6 [4].

Резистор со стороны коллектора ограничивает ток, протекающий в светодиоде ($I_{кэ}$ = 20 мА = 0.02 А), его номинал рассчитывается по формуле:

$R_к = (U_{vcc} - U_{vd} - \Delta U_{кэ}) / I_{кэ} = (5\text{-}2.5\text{-}0.1) / 0.02 = 120$ Ом

где U_{vcc} — напряжение питания;

U_{vd} — падение напряжения на светодиоде;

$\Delta U_{кэ}$ — падение напряжения на переходе коллектор-эммитер;

$I_к$ — предельный ток в цепи коллектора (ток через светодиод).

Номинал резистора со стороны базы рассчитывается по формуле:

$R_б = (U_{vcc} - \Delta U_{бэ}) / I_б = (U_{vcc} - \Delta U_{бэ}) / (I_{кэ} / \beta) = 4.4 / (0.02 / 200) = 44000$ Ом

где $\Delta U_{бэ}$ — падение напряжения на переходе база-эммитер;

$I_б$ — предельный ток в цепи базы;

β — коэффициент усиления транзистора

Ближайшее номинальное значение — 47 кОм.

Так как светодиоды, используемые в ССИ [5] имеют схожие характеристики, для управления ССИ были использованы транзисторные ключи с теми же номиналами резисторов.

Расчет транзисторного ключа для зуммера ($I_{кэ}$ = 20 мА) [6]:

$R_к = (U_{vcc} - U_{ha} - \Delta U_{кэ}) / I_{кэ} = (5\text{-}2\text{-}0.1) / 0.02 = 145$ Ом

Ближайшие стандартные номиналы резисторов — 143 и 150 Ом. Так как резистор 143 Ом дает ток выше требуемого, что может вывести из строя зуммер, был выбран номинал 150 Ом.

Тогда $I_к = (U_{vcc} - U_{ha} - \Delta U_{кэ}) / R_к^* = 2.9 / 150 = 19.3$ мА

$R_б = (U_{vcc} - \Delta U_{бэ}) / I_б = (U_{vcc} - \Delta U_{бэ}) / (I_{кэ} / \beta) = 4.4 / (0.0193 / 200) = 45595$ Ом

Ближайшее номинальное значение — 47 кОм.

1.2.3 Драйвер USB-UART (FT232RL)

Так как USB является одним из наиболее популярных интерфейсов подключения периферийных устройств к ПЭВМ, было решено осуществлять пересылку данных на ПЭВМ через него.

Так как МК ATmega8535 не имеет встроенной аппаратной поддержки USB, пересылка данных осуществляется через драйвер USB-UART. В качестве конкретной модели выбран драйвер FT232RL [7].

Структурная схема данного драйвера показана на рисунке 7.

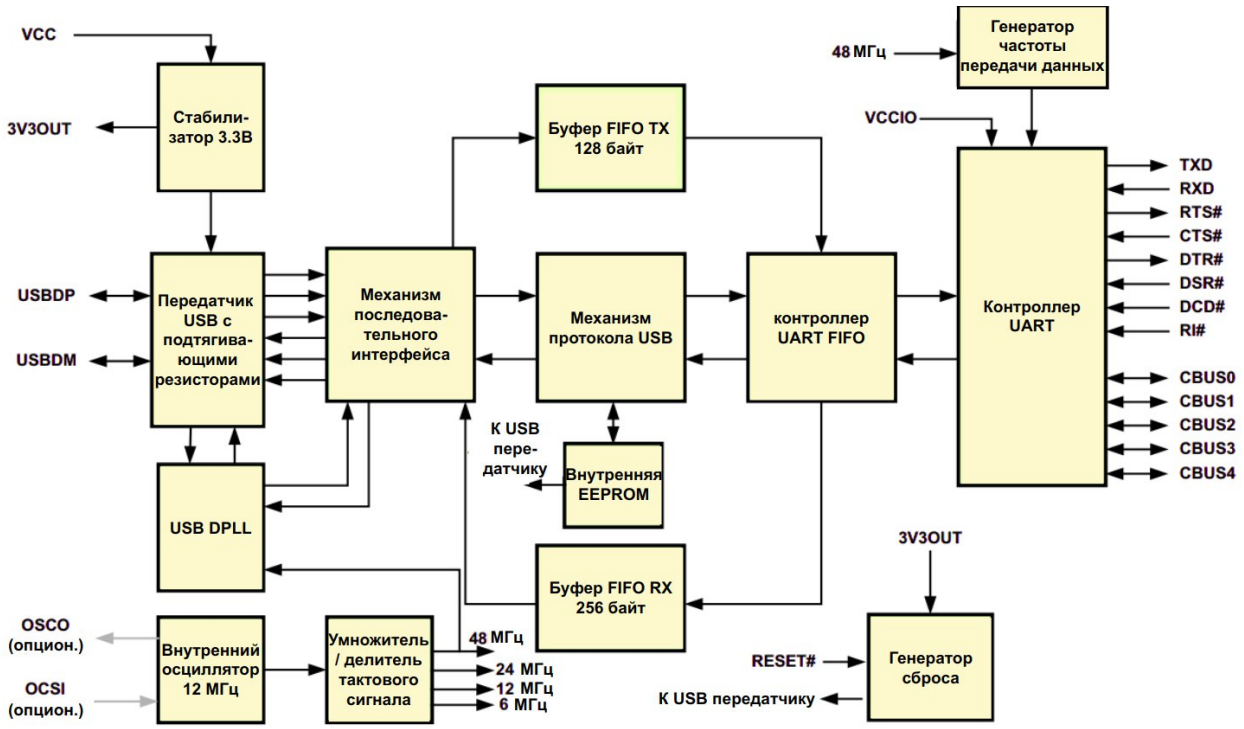

Рисунок 7 — структурная схема FT232RL.

Драйвер был подключен по схеме с питанием от шины, приведенной в документации. Схема представлена также на рисунке 8.

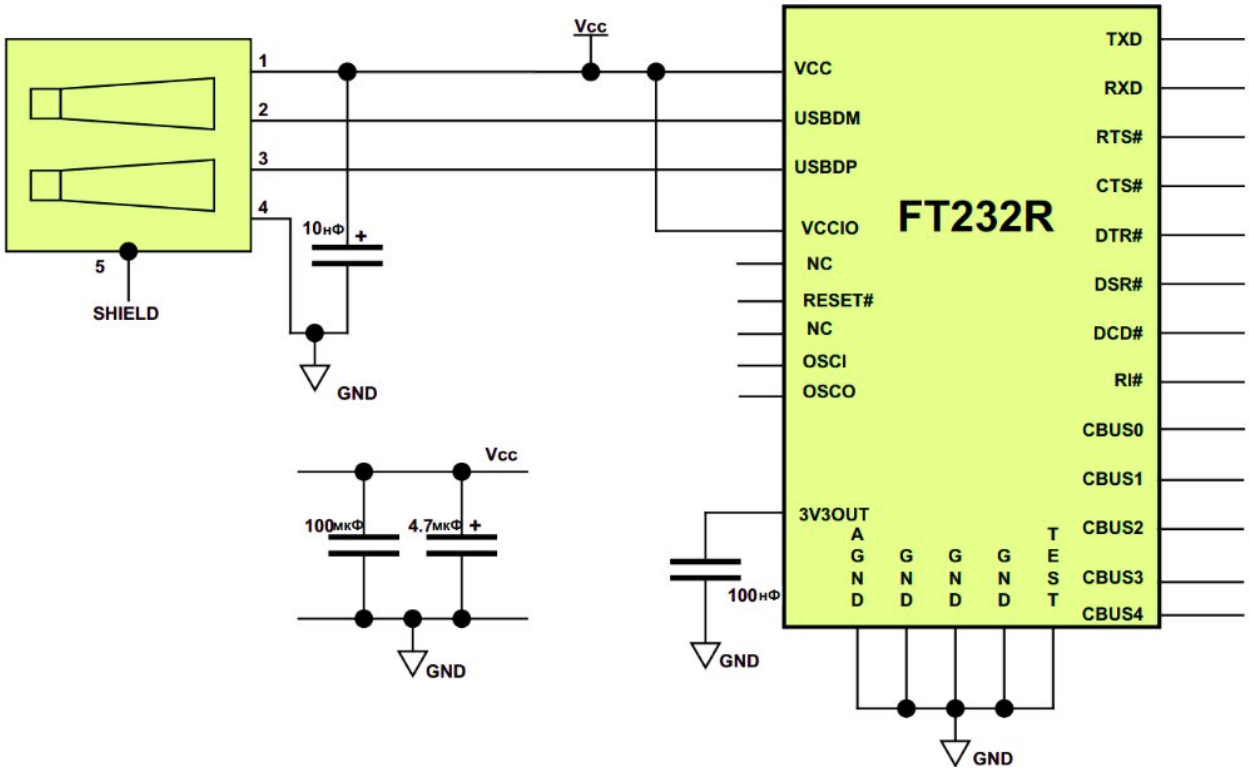

Рисунок 8 — схема подключения драйвера с питанием от шины

1.3 Расчет потребляемой мощности

Мощность, потребляемая одним устройством в статическом режиме рассчитывается по формуле P = U*I. Значения токов и напряжений на светодиодах, транзисторах и резисторах рассчитаны в разделе «Расчет транзисторных ключей». В качестве токов потребления микросхем взяты максимальные значения из документации, напряжение питания — всегда стандартное (5 В).

Расчет потребляемой мощности приведен в таблице 1.

Таблица 1 — потребляемая мощность

Элемент	Ток потребления, мА	Напряжение, В	Потребляемая мощность, мВт	Количество, шт	Суммарная потребляемая мощность, мВт
ATmega8535	40.0	5.0	200.0	1	200.0
FT232RL	15.0	5.0	75.0	1	75.0
SN74LS48	25.0	5.0	125.0	1	125.0
74HC137	8.0	5.0	40.0	1	40.0
CD4555	10.0	5.0	50.0	1*	50.0
CF-100 ($R_{к\_инд}$)	20.0	2.4	48.0	8*	384.0
L-865/4IDT (светодиодная полоса)	20.0	2.5	50.0	1*	50.0
FYQ-5641AUG-21 (7-сегментный дисплей)	20.0	2.5	50.0	7*	350.0

Продолжение таблицы 1

CF-100 ($R_{к\_зуммер}$)	19.3	2.9	55.97	1	55.97
KPI-G4214L (зуммер)	19.3	2.0	38.6	1	38.6

* — Для цепей светодиодных индикаторов учтено число максимально активных одновременно индикаторов (1 — для светодиодной матрицы и 7 — для дисплея).

Примечание: мощность, рассеиваемая на транзисторах, так как она пренебрежима мала по сравнению с мощностью, рассеиваемой на резисторах и светодиодах.

Суммарная потребляемая мощность устройства 1368.57 мВт.

1.4 Разработка программной части

1.4.1 Использованные библиотеки и подпрограммы

При написании программы была использована библиотека avr-libc, которая является адаптацией стандартной библиотеки Си под архитектуру микроконтроллеров AVR [8].

В проекте были указаны следующие заголовочные файлы из avr-libc:

- avr/io.h — содержит основную информацию, такую как имена регистров и констант;
- avr/interrupt.h — содержит адреса векторов прерываний;
- util/delay.h — позволяет управлять задержками;
- util/setbaud.h — упрощает управление скоростью передачи по UART.

Кроме того, был написан ряд собственных модулей и функций, позволяющих упростить управление подключенными к МК устройствами. Ниже приведены главные из них:

- uart.h — работа с UART:

- void uart_init() — инициализировать настройки UART;
- void uart_send_byte(char c) — отправить по UART один байт;
- void uart_send_long(long data) — преобразовать целое число в ASCII и отправить по UART;
- void uart_send_data(long* res_array, char range, long average) — отправить по UART данные о результатах испытаний.

- display.h — работа с дисплеем из ССИ:
  - void display_set_bytes(char t1, char t2, char t3, char t4) — сохранить значения разрядов в буферные переменные дисплея;
  - void display_set_long(long target) — разбить целое число на разряды и сохранить в буферных переменных дисплея;
  - void display_flash_once() — переключить активный индикатор дисплея;
  - void display_off() — выключить дисплей;
- leds.h — управление светодиодной матрицей;
  - char leds_random_line() — присвоить переменной, хранящей номер активной строки случайное значение из интервала [0;8);
  - void leds_move_column() — изменить номер активного столбца;
  - void leds_off() — погасить светодиоды.
- keyboard.h — работа с клавиатурой:
  - char keyboard_get_state() — получить номер нажатой клавиши.

С помощью режима отладки и показателя Stop Watch было подсчитано время исполнения некоторых из перечисленных выше процедур. Результаты измерений показаны в таблице 2.

Таблица 2 — время исполнения процедур

Процедура	$t_{нач}$, мкс	$t_{кон}$, мкс	Δt, мкс
uart_send_byte(char c)	726.50	729.25	2.75
uart_send_long(long data)	729.25	1776.50	1047.25
display_set_long(long target)	1776.50	2882.0	1105.5
display_flash_once()	2882.0	2887.75	5.75
leds_random_line()	2887.75	3310.50	422.75
keyboard_get_state()	3315.50	3328.75	13.25

1.4.2 Алгоритм основной программы

Схема алгоритма основной программы представлена на рисунках 9 и 10, а исходный код основной программы — в приложении А.

Основная идея алгоритма состоит в том, что система может находиться либо в режиме прохождения серии испытаний, либо в режиме паузы.

В активном режиме (режиме прохождения испытаний) можно ускорять или замедлять скорость движения мишени. В случае правильного «захвата цели» на дисплей будет выведено время реакции в данном испытании в миллисекундах, а горящий светодиод «переместится» на другую строку матрицы. Иначе — система издаст звуковой сигнал. После окончания заданного числа испытаний система выведет среднее время реакции и перейдет в режим паузы. Также испытания можно прервать нажатием кнопки «Restart» (в этом случае дисплей будет погашен).

В режиме паузы светодиоды погашены. Если серия испытаний была пройдена до конца, можно отправить на ПЭВМ через драйвер UART-USB информацию о времени реакции на каждую мишень и о среднем времени реакции. Можно начать новое испытание кнопкой «Restart», переменные для

подсчета времени и числа пройденных испытаний обнулятся, как и показания дисплея.

Назначение некоторых используемых переменных и функций:

- active — флаг режима прохождения испытаний;
- timer_ms — время с начал испытания (обновляется по прерыванию от T1);
- tries_counter — счетчик числа завершенных испытаний в серии;
- void display_set_long(long target); — вывести на дисплей значение, либо специальные символы, если значение не умещается;
- char leds_random_line(); — «перемещает» мишень в другую строку и возвращает номер строки.

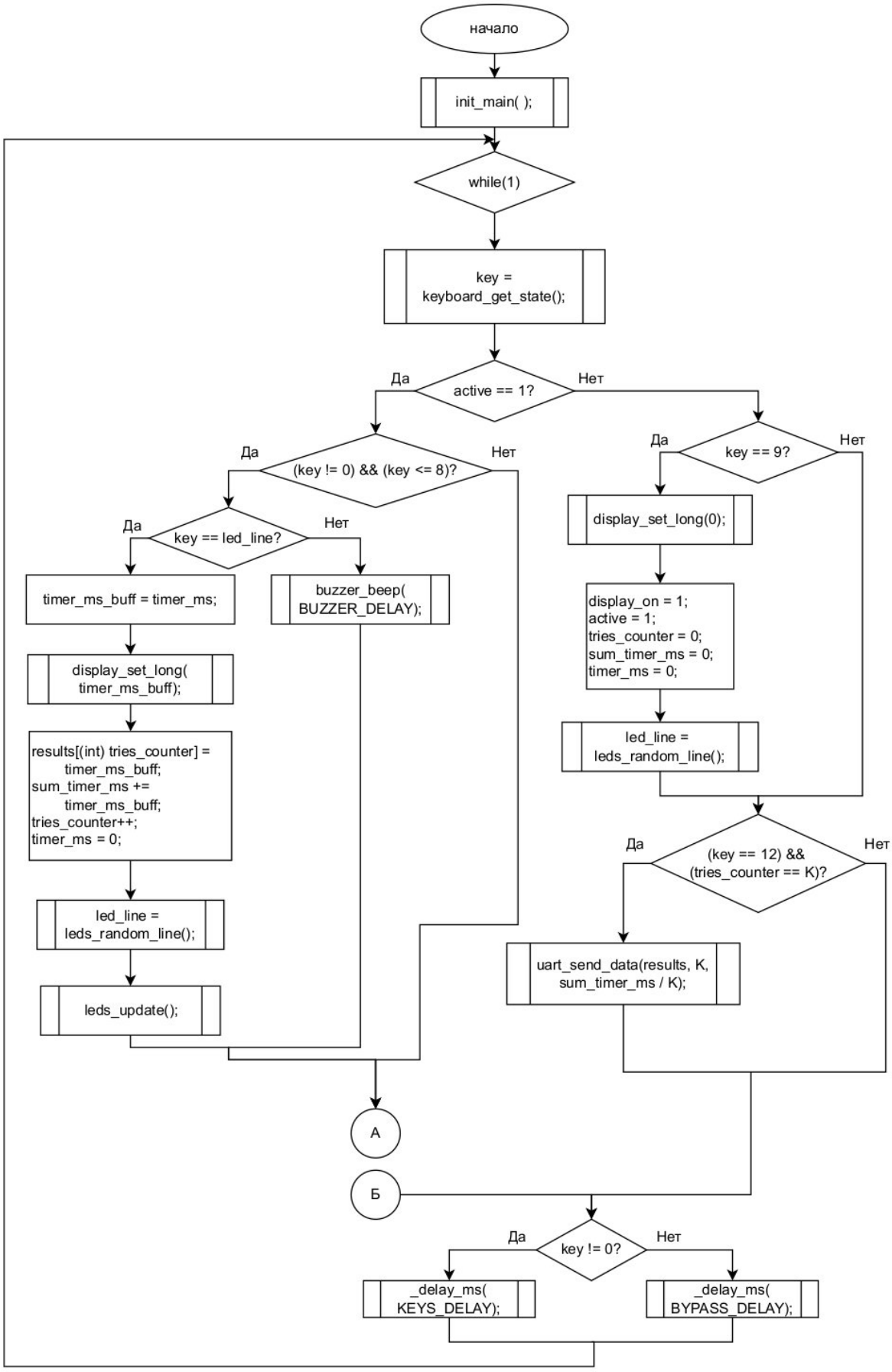

Рисунок 9 — алгоритм основной программы

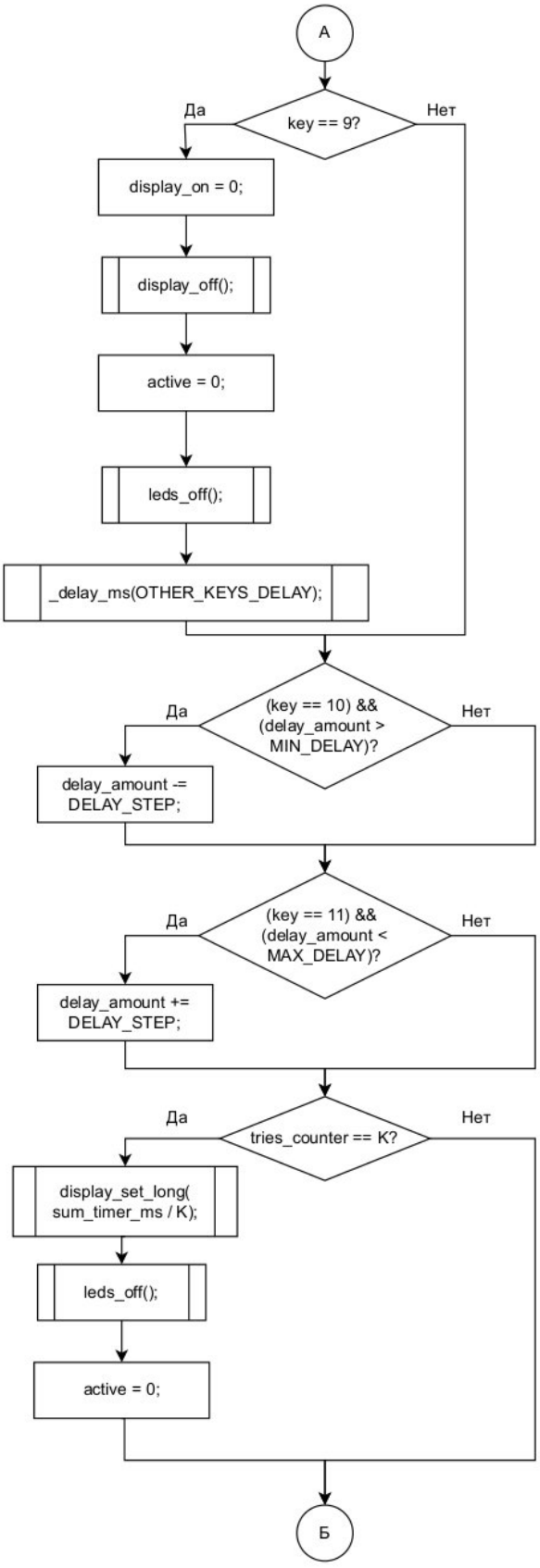

Рисунок 10 — алгоритм основной программы

1.4.3 Управление динамической индикацией

Для отображения скорости реакции используется дисплей из четырех ССИ. На нем осуществляется динамическая индикация числа с переключением разрядов по переполнению таймера T0.

Минимальная частота при которой человек воспринимает мигающее изображение как статическое — 24 Гц. Так как используется четыре индикатора, их необходимо переключать с частотой 24*4 = 96 Гц. При тактовой частоте 4 МГц и размерности таймера в 8 бит коэффициент предделителя можно определить из уравнения:

$4\ 000\ 000 / (2^8 * K) = 24 * 4$

$K = 4\ 000\ 000 / (2^8 * 24 * 4) = 162.76$

Наиболее близкие значения предделителя к полученному — 64 и 256. Но выбор значения 256 приведет к тому, что частота индикации будет ниже предельной. Поэтому было выбрано значение 64. Для установки этого значения в соответствии с таблицей 3 были установлены биты CS01 и CS0 регистра TCCR0.

Таблица 3 — выбор источника тактового сигнала для таймера/счётчика T0

CS02	CS01	CS00	Источник тактового сигнала
0	0	0	Таймер/счётчик остановлен
0	0	1	CK (тактовый сигнал микроконтроллера)
0	1	0	CK/8
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	Вывод T0, инкремент счётчика производится по спадающему фронту импульсов
1	1	1	Вывод T0, инкремент счётчика производится по нарастающему фронту импульсов

1.4.4 Отсчет времени реакции

Отсчет времени реакции реакции производится за счет переменной timer_ms, инкрементируемой каждую миллисекунду по прерыванию таймера T1, работающего в режиме сравнения.

Кроме того, в обработчике этого прерывания можно осуществлять переключение горящего светодиода в светодиодной матрице, чтобы организовать задержку переключения светодиодов независимо от задержки считывания кнопок.

Для повышения точности возьмем значение предделителя K=1. Рассчитаем значение регистра сравнения OCR1A:

(F_CPU / K) / OCR1A = △t

Где △t = 1c = 1000 мс, K = 1

(F_CPU / 1) / OCR1A = 1000

OCR1A = F_CPU / 1000

Код обработчика прерываний T1 в режиме сравнения приведен в листинге 1.

Листинг 1 — код обработчика прерываний T1 в режиме сравнения

```
ISR (TIMER1_COMPA_vect)
{
 timer_ms++;
 if((active == 1) && (timer_ms % delay_amount == 0)){
        leds_move_column();
        leds_update();
 }
 TCNT1=0; //clear ticks
}
```

С помощью режима отладки и показателя Stop Watch было определено время обработки прерывания при выполнении условий ветвления и при их невыполнении.

Данный процесс проиллюстрирован на рисунках 11-13.

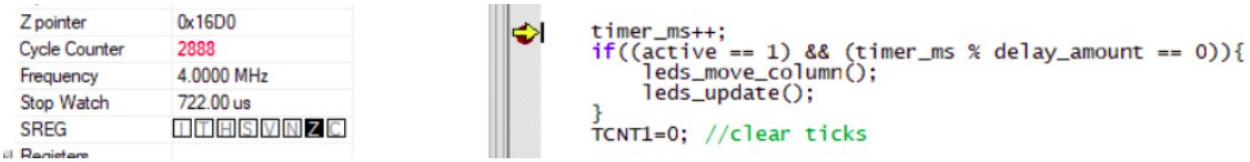

Рисунок 11 — начало обработки прерывания

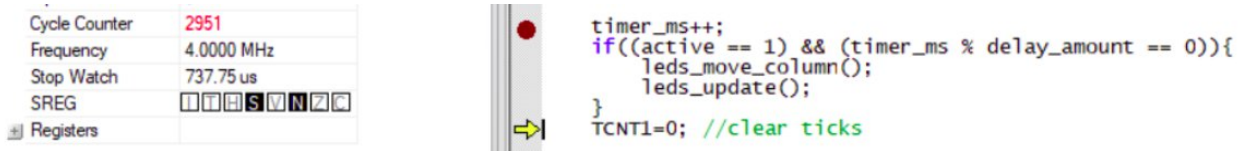

Рисунок 12 — конец обработки прерывания при несоблюдении условий ветвления

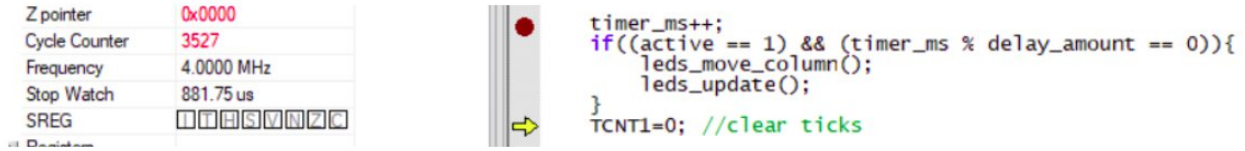

Рисунок 13 — конец обработки прерывания при соблюдении условий ветвления

Время обработки прерывания в случае несоблюдения и соблюдения условий ветвления составило соответственно:

t_{false} = (737.75 - 722.00) = 15.75 мкс

t_{true} = (881.75 - 722.00) = 159.75 мкс

Очевидно, что при минимальном времени задержки между переключениями светодиодов будет получено максимальное значение времени обработки прерывания, а значит и максимальная погрешность значения timer_ms.

Минимальное время задержки между переключениями светодиодов (MIN_DELAY) составляет 100 мс = 100 000 мкс. Отсюда было найдено значение относительной погрешности timer_ms.

$\delta t = (\Delta t / t) * 100\% = ((99 * t_{false} + t_{true}) / t) * 100\% = ((99 * 15.75 + 159.75) / 100\ 000) * 100\% = 1.719\%$.

При скорости реакции в пределах 10 с (что крайне много) погрешность не превысит 18 мс. По сравнению с задержкой считывания с клавиатуры и временем выполнения цикла основной программы это число незначительно

1.4.5 Работа с UART

В качестве способа передачи информации на ПЭВМ в разработанном устройстве используются интерфейсы UART и USB, связь которых осуществляется через соответствующий драйвер.

Кадр UART имеет формат, приведенный на рисунке 14.

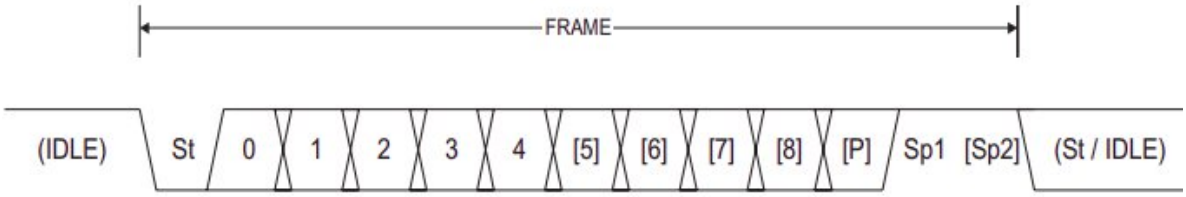

Рисунок 14 — формат кадра UART

Структура модуля UART показана на рисунке 15.

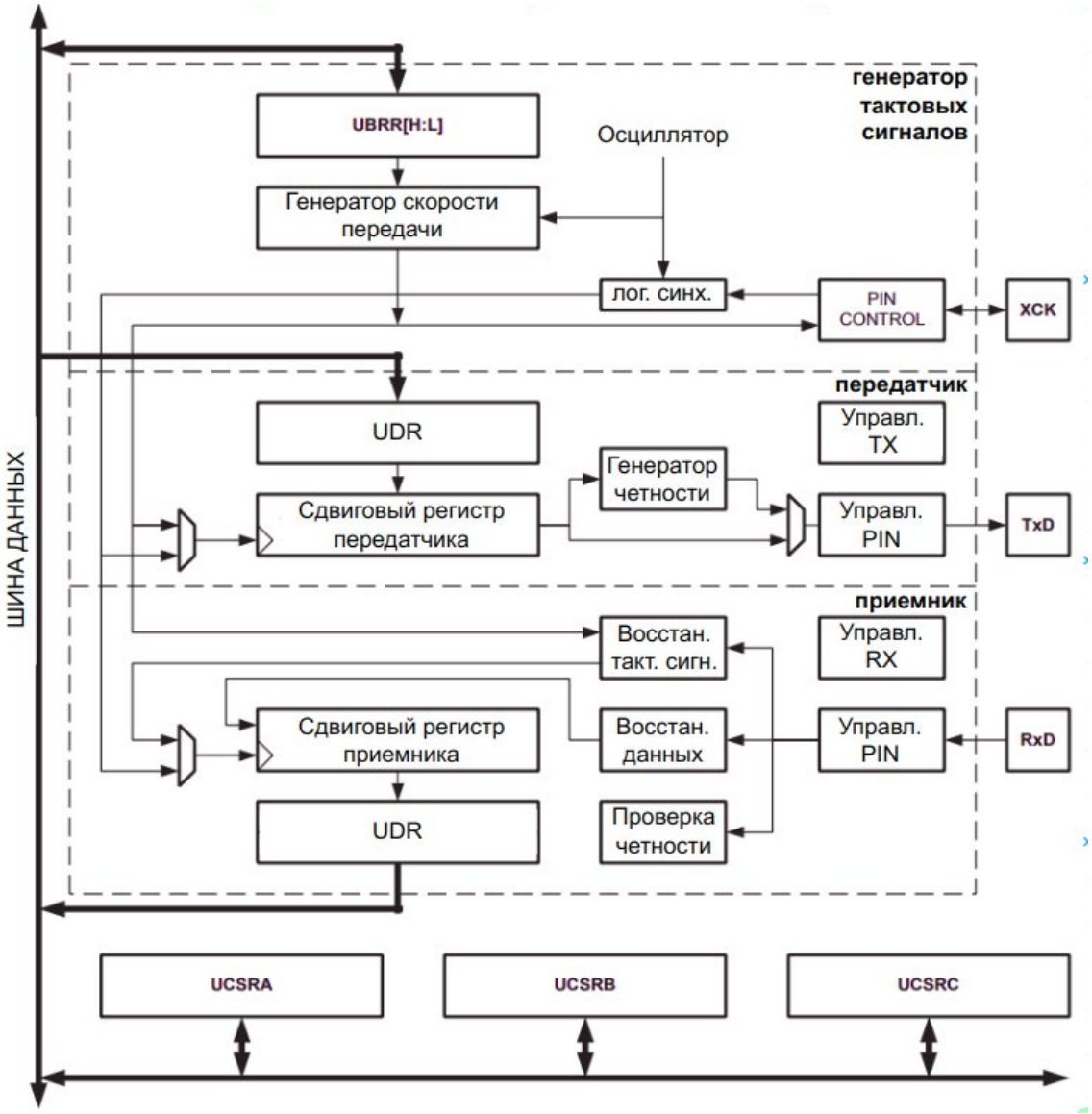

Рисунок 15 — структура модуля UART

St — стартовый бит (всегда на уровне логического 0);

0-8 — биты данных;

P — бит четности;

Sp1, Sp2 — стоповые биты (всегда на уровне логической 1);

IDLE — передачи данных не происходит (уровень логической 1 на линии).

Управление работой приёмопередатчика осуществляется с помощью регистров управления UCSRB и UCSRC. Текущее состояние приёмопередатчика определяется с помощью регистра состояния UCSRA. Формат этих регистров приведен в таблицах 4,5,7.

Таблица 4 — Формат регистра управления UCSRB

№ разряда	7	6	5	4	3	2	1	0
Имя	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCZ2	RXB8	TXB8

Назначение управляющих битов:

RXCIE — разрешение прерывания по завершении приёма;

TXCIE — разрешение прерывания по завершении передачи;

UDRIE — разрешение прерывания при опустошении регистра данных UART;

RXEN — разрешение приёма;

TXEN — разрешение передачи;

UCZ2 — (вместе с UCZ0 и UCZ1) — количество полезных информационных битов;

RXB8 - 8-й разряд принимаемых данных.

TXB8 - 8-й разряд передаваемых данных.

Таблица 5 — Формат регистра управления UCSRC

№ разряда	7	6	5	4	3	2	1	0
Имя	URSEL	UMSEL	UPM1	UPM0	USBS	UCZ1	UCZ0	UCPOL

URSEL — выбор между UBRRH и UCSRC;

UMSEL — выбор синхронного режима работы;

UPM1 и UPM0 — режим проверки чётности;

USBS — определяет количество стоповых битов;

UCZ0 и UCZ1 (вместе с UCZ2) — количество полезных информационных битов;

UCPOL — выбор фронта синхросигнала (для синх. режима).

Выбор размера кадра осуществляется согласно таблице 6.

Таблица 6 — выбор размера кадра

UCZ2	UCZ1	UCZ0	Размер кадра
0	0	0	5 бит
0	0	1	6 бит
0	1	0	7 бит
0	1	1	8 бит
100 — 110			зарезервировано
1	1	1	9 бит

Таблица 7 — Формат регистра состояния UCSRA

№ разряда	7	6	5	4	3	2	1	0
Имя	RXC	TXC	UDRE	FE	DOR	PE	U2X	MCPM

RXC — флаг завершения приёма. Данный флаг устанавливается в «1» при пересылке принятого слова из сдвигового регистра приёмника в регистр данных UDR. Сбрасывается флаг аппаратно при чтении регистра UDR;

TXC — флаг завершения передачи. Данный флаг устанавливается в «1» после передачи всех разрядов слова, включая стоп-бит, из сдвигового регистра передатчика, при условии, что в регистр данных UDR не было загружено новое значение.

UDRE – регистр данных пуст. Данный флаг устанавливается в «1» после пересылки байта из регистра данных UDR в сдвиговый регистр передатчика. Установка этого флага означает, что передатчик готов к получению нового значения для передачи. Сбрасывается флаг аппаратно при записи в регистр UDR;

FE – флаг ошибки формата. Данный флаг устанавливается в «1», если стоп-бит принятого слова равен «0». Флаг сбрасывается при приёме стоп-бита, равного «1»;

DOR — флаг переполнения. Данный флаг устанавливается в «1», если в сдвиговом регистре приёмника находится новое принятое слово, а старое содержимое регистра UDR не прочитано. Флаг остаётся установленным до тех пор, пока не будет прочитано содержимое регистра UDR;

PE — флаг ошибки четности;

U2X — двоение скорости передачи;

MPCM — режим мультипроцессорного взаимодействия;

Управление скоростью приёма и передачи данных осуществляется контроллером скорости передачи, который является обыкновенным делителем частоты. Скорость передачи зависит от содержимого регистров UBRRH:UBRRL.

Скорость передачи определяется следующим выражением:

$BAUD = f_{clk} / (16 * (UBRR + 1))$

На основе описанного выше механизма настройки UART был написан программный драйвер, фрагмент исходного кода которого приведен в листинге 2.

Листинг 2 — программный драйвер UART

```
include <avr/io.h>
#define BAUD 9600 /* частота 9600 */
#include <util/setbaud.h> /* упрощенная настройка частоты */

void uart_init() {
   UBRRH = UBRRH_VALUE; /* упрощенная настройка частоты */
   UBRRL = UBRRL_VALUE;  /* упрощенная настройка частоты */

#if USE_2X
   UCSRA |= (1 << U2X);
#else
   UCSRA &= ~(1 << U2X);
#endif

   UCSRC = (1 << UCSZ1) | (1 << UCSZ0); /* 8-битный кадр */
   UCSRB = (1 << RXEN) | (1 << TXEN);   /* Разрешить получение и передачу */
}

void uart_send_byte(char c) {
  loop_until_bit_is_set(UCSRA, UDRE); /* Ждать опустошения регистра данных */
  UDR = c; /* Записать с в регистр данных (передать с на ПЭВМ *.
}
```

2 Технологическая часть

2.1 Тестирование программы

Для тестирования программы была построена упрощенная схема системы в среде Proteus 8 (рисунок 16).

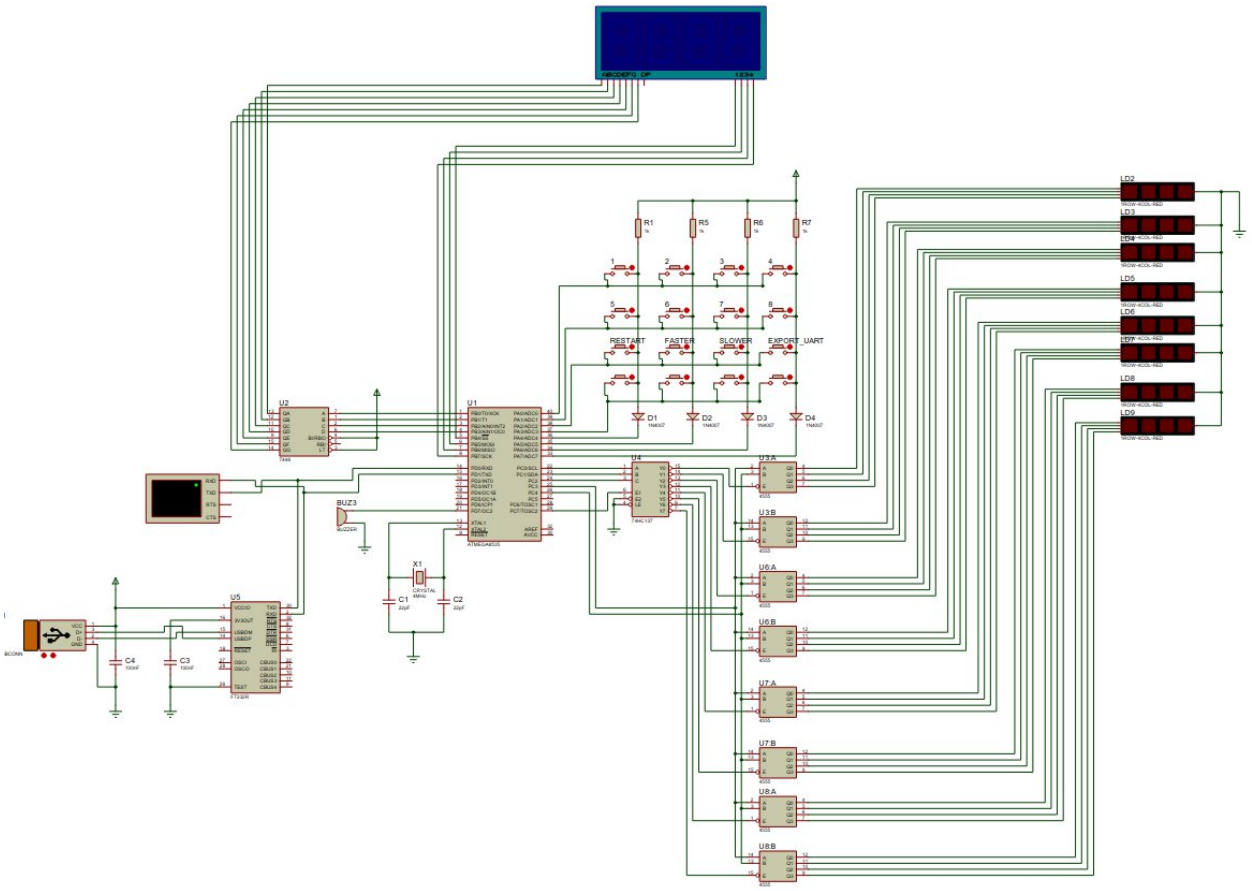

Рисунок 16 — упрощенная схема тренажера оператора в Proteus

Ввод информации систему осуществляется посредством матричной клавиатуры 4x4. Вывод информации осуществляется на светодиодную матрицу 8x4, дисплей из ССИ. Для проверки работы UART используется виртуальный терминал.

Между серий испытаний экран и светодиодная матрица погашены. В начале каждой новой серии испытаний дисплей отображает «0000». После каждого правильного выбора строки виртуальный дисплей отображает в миллисекундах время, за которое оператор отреагировал на мишень.

Если время реакции >9999 мс, то дисплей отображает специальные символы, напоминающие литеру «E» (что подразумевает «Error» — ошибка

отображения). При этом испытание все равно считается выполненным и его результаты будут сохранены в памяти МК.

Пример отображения времени реакции <10000 мс показан на рисунке 17.

Рисунок 17 — отображение времени реакции

Состояние дисплея при превышении его разрядности (время реакции >9999 мс) показано на рисунке 18.

Рисунок 18 — состояние дисплея при превышении разрядности

Пример передачи данных на ПЭВМ показан на рисунке 19. Для удобства тестирования было установлено число испытаний в серии К=3.

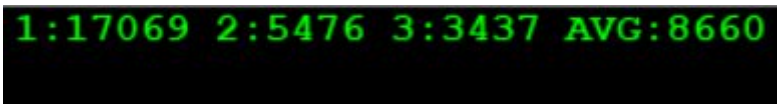

Рисунок 19 — передача данных по UART

Также было проверено, что МК издает звуковой сигнал при неправильном нажатии клавиши. Кроме того, проверено, что нельзя передать данные по UART пока не пройдена до конца серия испытаний, а также проверено выключение дисплея при прерывании серии испытаний кнопкой «RESTART» и установка «0000» при запуске новой серии испытаний. Проверена работоспособность кнопок регулирования скорости движения светодиодов.

2.2 Программирование микроконтроллера

В составе микроконтроллеров AVR наряду с Flash-памятью программ имеется энергонезависимая память для хранения данных EEPROM, которая также может быть запрограммирована перед началом работы. Микроконтроллеры этого семейства допускают более широкие возможности при программировании и верификации.

В процессе программирования микроконтроллеров AVR осуществляют:

- запись команд и констант в НазН-память программ;
- запись данных в память EEPROM;
- запись конфигурационных битов (fuse bits);
- запись битов защиты (lock bits).

В ходе этих работ можно дополнительно выполнить следующие операции:

- стирание памяти микроконтроллера;
- чтение ячеек Flash-памяти программ и памяти данн EEPROM для контроля правильности записанной информации (верификация);
- чтение конфигурационных ячеек для определения состояния микроконтроллера;
- чтение битов защиты;
- чтение ячеек идентификатора для определения типа микроконтроллера и ячейки калибровочного байта при настройке внутреннего RC-генератора.

Программирование микроконтроллеров AVR в зависимости от применяемого класса и типа модели может быть выполнено четырьмя способами:

- последовательное программирование при высоком напряжении питания (+12 В);
- параллельное программирование при высоком напряжении;

– последовательное программирование при низком напряжении по интерфейсу SPI;
– программирование по интерфейсу JTAG.

Способы 2 и 3 поддерживаются всеми микроконтроллерами семейства Mega и большинством микроконтроллеров семейства Classic, по интерфейсу JTAG — микроконтроллерами ATmega16x/162x/323x/64x/128x семейства Mega.

Режим последовательного программирования по интерфейсу SPI используется, как правило, для программирования (перепрограммирования) микроконтроллера непосредственно в системе (In System Programming, ISP). Особенно широко применяется в изделиях, представляющих стартовые наборы разработчика, используемые в процессе обучения.

Схема подключения микроконтроллера при программировании по интерфейсу SPI показана на рисунке 20.

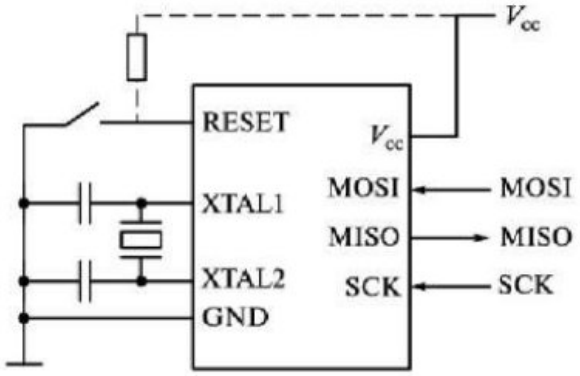

Рисунок 20 — схема подключения микроконтроллера при программировании по SPI

Программирование по интерфейсу SPI осуществляется путем посылки четырехбайтовых команд на вывод MOSI микроконтроллера, в которых один или два байта определяют тип операции, остальные — адрес, записываемый байт, установочные биты и биты защиты, холостой байт. При выполнении операции чтения считываемый байт снимается через вывод MISO. Передача команд и вывод результатов их выполнения осуществляется от старшего разряда к младшему. При этом «защелкивание» входных данных

выполняется по нарастающему фронту сигнала SCK, а «защелкивание» выходных данных — по спадающему.

Программирование памяти программ микроконтроллеров семейства Mega осуществляется постранично. Сначала содержимое страницы побайтно заносится в буфер по командам «Загрузка страницы Flash-памяти». При этом в каждой команде указываются младшие разряды адреса записываемой ячейки (положение ячейки на странице) и записываемое значение. Каждая ячейка памяти загружается в следующей последовательности: сначала младший байт, затем — старший. Фактическое программирование страницы Flash-памяти выполняется после завершения загрузки буфера по команде «Запись страницы Flash-намяти». В команде указываются старшие разряды адреса, определяющие номер страницы.

Программирование памяти данных EEPKOM осуществляется с помощью команд «Запись в память EEPROM». В каждой команде указывается адрес записываемой ячейки и записываемое значение. Для большинства микроконтроллеров можно также программировать конфигурационные биты (fuse) для выбора настроек тактирования, длительности задержки при старте и порога детектора понижения напряжения (BOD).

Команды, используемые для программирования ATmega8535 показаны в таблице 8.

Таблица 8 — команды программирования ATmega8535

Инструкция	Формат инструкции			
	Байт 1	Байт 2	Байт 3	Байт 4
Разрешить программирование	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx
Стереть чип	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx
Прочитать из памяти программ	0010 H000	0000 aaaa	bbbb bbbb	oooo oooo

Продолжение таблицы 8

Загрузить в страницу памяти программ	0100 H000	0000 xxxx	xxxb bbbb	iiii iiii
Записать страницу памяти программ	0100 1100	0000 aaaa	bbbx xxxx	xxxx xxxx
Прочитать из памяти EEPROM	1010 0000	00xx xxxa	bbbb bbbb	oooo oooo
Записать в память EEPROM	1100 0000	00xx xxxa	bbbb bbbb	iiii iiii
Прочитать биты защиты	0101 1000	0000 0000	xxxx xxxx	xxoo oooo
Записать биты защиты	1010 1100	111x xxxx	xxxx xxxx	11ii iiii
Чтение сигнатурного бита	0011 0000	00xx xxxx	xxxx xxbb	oooo oooo
Записать конфигурационные биты	1010 1100	1010 0000	xxxx xxxx	iiii iiii
Записать старшие конфигурационные биты	1010 1100	1010 1000	xxxx xxxx	iiii iiii
Прочитать конфигурационные биты	0101 0000	0000 0000	xxxx xxxx	oooo oooo
Прочитать старшие конфигурационные биты	0101 1000	0000 1000	xxxx xxxx	oooo oooo

Продолжение таблицы 8

Чтение калибровочного байта	0011 1000	00xx xxxx	0000 00bb	oooo oooo

Обозначения:

- a — адрес старших разрядов;
- b — адрес младших разрядов;
- h — 0 – младший байт, 1 – старший байт;
- o — вывод данных;
- i — ввод данных;
- x — произвольное значение.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было спроектировано устройство для измерения времени реакции оператора на движущуюся мишень. Устройство может выводить информацию о результатах испытаний на дисплей из ССИ и пересылать эти данные на ПЭВМ через интерфейс USB.

Для написания исходного кода была использована среда AVR Studio 4, компилятор GCC, а так же библиотека AVR Libc. Кроме того, было произведено моделирование системы в среде Proteus 8.

Был подготовлен набор документации на объект разработки, состоящий из расчетно-пояснительной записки, функциональной схемы, принципиальной схемы, листинга программного кода, спецификации радиоэлементов, использованных в системе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хартов В.Я. Методические указания к выполнению курсовой работы по дисциплине «Микропроцессорные системы»: учебно-методическое пособие / В.Я. Хартов. — Москва: Издательство: МГТУ им. Н.Э. Баумана, 2021. – 31 с.
2. ATmega8535 Datasheet [Электронный ресурс]. – Режим доступа: https://static.chipdip.ru/lib/059/DOC000059794.pdf (дата обращения: 15.09.2022)
3. SN74LS48 Datasheet [Электронный ресурс]. – Режим доступа: https://datasheetspdf.com/pdf-file/1089745/Motorola/SN74LS48/1 (дата обращения: 25.09.2022)
4. Diodov.net: Транзисторный ключ [Электронный ресурс]. – Режим доступа: https://diodov.net/tranzistornyj-klyuch/ (дата обращения: 10.10.2022)
5. FYQ-5641AUG-21 Datasheet [Электронный ресурс]. – Режим доступа: https://files.rct.ru/pdf/indicator/fyq-5641ax_bx.pdf (дата обращения: 10.10.2022)
6. KPI-G4214L Datasheet [Электронный ресурс]. – Режим доступа: https://files.rct.ru/pdf/buzzer/kpi-g4000p.pdf (дата обращения: 16.10.2022)
7. FT232RL Datasheet [Электронный ресурс]. – Режим доступа: https://static.chipdip.ru/lib/222/DOC000222844.pdf (дата обращения: 26.10.2022)
8. AVR-Libc: Online User Manual [Электронный ресурс]. – Режим доступа: https://www.nongnu.org/avr-libc/user-manual/index.html (дата обращения: 05.11.2022)

ПРИЛОЖЕНИЕ А

Исходный текст программы

main.c

```
#define F_CPU 4000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#include "leds.h"
#include "keyboard.h"
#include "uart.h"
#include "display.h"
#include "buzzer.h"
#include "timers.h"

#define BUZZER_DELAY 50
#define DEFAULT_DELAY 200
#define MIN_DELAY 100
#define MAX_DELAY 500
#define DELAY_STEP 10
#define KEYS_DELAY 500
#define BYPASS_DELAY 10

#define K 3

long debug_activate_ms;

char display_on = 0;
char key = 0;
char led_line = 0;
char active = 0;
long timer_ms = 0;
long timer_ms_buff;
long timer_control; // for debug
int delay_amount = DEFAULT_DELAY;

char tries_counter = 0;
long sum_timer_ms = 0;

long results[K];

ISR (TIMER1_COMPA_vect)
{
```

```
  timer_ms++;
  if((active == 1) && (timer_ms % delay_amount == 0)){
        leds_move_column();
        leds_update();
  }
  TCNT1=0; //clear ticks
  timer_control++; // for debug
}

ISR (TIMER0_OVF_vect){
        if(display_on == 1){
                display_flash_once();
        }
}

void init_main(void){
        timer0_init();
        timer1_init();
        DDRA=0xF0;
        PORTA=0x0F; // resistors on buttons
        DDRC=0xFF; // PORTC - OUTPUT
        DDRB=0xFF; // PORTB - OUTPUT
        DDRD=0xFF; // PORTD - OUTPUT
        uart_init();
        sei();

        display_off();
        led_line = leds_random_line();
        display_set_long(0);
}

int main(){
        init_main();

        while(1){
                key = keyboard_get_state();

                if(active == 1){

                        if((key != 0) && (key <= 8)){
                                if(key == (led_line + 1)){
                                        timer_ms_buff = timer_ms;
                                        display_set_long(timer_ms_buff);
                                        results[(int) tries_counter] = timer_ms_buff;
                                        sum_timer_ms += timer_ms_buff;
```

```
                        tries_counter++;
                        timer_ms = 0;
                        led_line = leds_random_line();
                } else {
                        buzzer_beep(BUZZER_DELAY);
                }
        }

        if(key == 9){
                display_on = 0;
                display_off();
                active = 0;

                leds_off();
        }

        if((key == 10) && (delay_amount > MIN_DELAY)){
                delay_amount -= DELAY_STEP;
        }

        if((key == 11) && (delay_amount < MAX_DELAY)){
                delay_amount += DELAY_STEP;
        }

        if(tries_counter == K){
                display_set_long(sum_timer_ms / K);
                leds_off();
                active = 0;
        }

} else {

        if(key == 9){
                display_set_long(0);
                display_on = 1;
                active = 1;

                tries_counter = 0;
                sum_timer_ms = 0;
                timer_ms = 0;
                led_line = leds_random_line(); // refresh line after reset
        }

        if((key == 12)&&(tries_counter == K)){
                        uart_send_data(results, K, sum_timer_ms / K);
```

```
                        }
                }

                if(key != 0){
                        _delay_ms(KEYS_DELAY);
                } else {
                        _delay_ms(BYPASS_DELAY);
                }
        }

        return 0;
}
```

Uart.h

```
#ifndef UART_H
#define UART_H

void uart_init();
void uart_send_byte(char target);
void uart_send_long(int data);
void uart_send_data(long* res_array, char range, long average);

#endif
```

Uart.c

```
#include <avr/io.h>
#define BAUD 9600
#include <util/setbaud.h>

long divider;
char cnt;
char i;

void uart_init() {
   UBRRH = UBRRH_VALUE;
   UBRRL = UBRRL_VALUE;

#if USE_2X
   UCSRA |= (1 << U2X);
#else
   UCSRA &= ~(1 << U2X);
#endif
```

```
   UCSRC = (1 << UCSZ1) | (1 << UCSZ0); // 8-bit data
   UCSRB = (1 << RXEN) | (1 << TXEN);   // Enable RX and TX
}

void uart_send_byte(char c) {
  loop_until_bit_is_set(UCSRA, UDRE); // Wait until data register empty. */
  UDR = c;
}

void uart_send_long(long data) {
        divider = 1;
        while((divider*10) < data){
                divider *= 10;
        }
        while(divider > 0){
                uart_send_byte(((data / divider) % 10) + '0');
                divider /= 10;
        }
        uart_send_byte(' ');
}

void uart_send_data(long* res_array, char range, long average) {
        for(i=0; i<range; i++){
                uart_send_byte(i+1+'0');
                uart_send_byte(':');
                uart_send_long(res_array[(int) i]);
                uart_send_byte('\n');
        }
        uart_send_byte('A');
        uart_send_byte('V');
        uart_send_byte('G');
        uart_send_byte(':');
        uart_send_long(average);
        uart_send_byte('\n');
}
```

Leds.h

```
#ifndef LEDS_H
#define LEDS_H

char leds_random_line();
void leds_move_column();
void leds_update();
```

```
void leds_off();

#endif
```

Leds.c

```
#include <stdlib.h>
#include <avr/io.h>

char line = 0;
char column = 0;
char direction = 1; // forward

char leds_random_line(){
	line = rand() % 8;
	return line;
}

void leds_update(){
	PORTC = 0x80 | (column << 3) | line;
}

void leds_move_column(){
	if(direction == 1){
		if(column < 3){
			column++;
		} else {
			direction = 0;
			column--;
		}
	} else {
		if(column > 0){
			column--;
		} else {
			direction = 1;
			column++;
		}
	}
}

void leds_off(){
	PORTC &= 0x7F;
}
```

Keyboard.h

```
#ifndef KEYBOARD_H
#define KEYBOARD_H

char keyboard_get_state();

#endif
```

Keyboard.c

```
#include <avr/io.h>

char i=0,j=0;
char portState[4]= {0xEF,0xDF,0xBF,0x7F};
char inputState[4]={0x01,0x02,0x04,0x08};

/*
        returns (line<<4 | coluumn)
        values in [0;3]
        j - line
        i - column
*/
char keyboard_get_state(){
        for(i=0; i<4; i++)
  {
    PORTA=portState[(int) i];
    for(j=0; j<4; j++)
    {
      if(((PINA&inputState[(int) j])==0))
      {
                        return (j*4 + i + 1);
      }
    }
  }
        return 0;
}
```

Display.h

```
#ifndef DISPLAY_H
#define DISPLAY_H
```

```
void display_set_bytes(char t1, char t2,
        char t3, char t4);

void display_flash_once();
void display_off();
void display_set_long(long target);

#endif
```

Display.c

```
#include <avr/io.h>
#include <util/delay.h>

char c[4];
char d1, d2, d3, d4;
char display_pos = 0;

void display_set_bytes(char t1, char t2,
        char t3, char t4){
        c[0] = ~0x1F | (0x0F & t1);
        c[1] = ~0x2F | (0x0F & t2);
        c[2] = ~0x4F | (0x0F & t3);
        c[3] = ~0x8F | (0x0F & t4);
}

void display_set_long(long target){
        if(target < 10000){
                d4 = target % 10;
                d3 = (target / 10) % 10;
```

```
            d2 = (target / 100) % 10;
            d1 = (target / 1000) % 10;
            display_set_bytes(d1, d2, d3, d4);
        } else {
            display_set_bytes(0x0E, 0x0E, 0x0E, 0x0E);
        }
}

void display_off(){
        PORTB |= 0xFF;
}

void display_flash_once(){
        display_pos++;
        if(display_pos > 3){
            display_pos = display_pos % 4;
        }
        PORTB = c[(int) display_pos];
}
```

Timers.h

```
#ifndef TIMERS_H
#define TIMERS_H

void timer0_init();
void timer1_init();

#endif
```

Timers.c

```
#include <avr/io.h>

void timer0_init(){
	//TCCR0 = (1<<CS01) | (1 << CS00); // K = 64
	TCCR0 = (1<<CS01); // K = 8
	TIMSK |= (1<<TOIE0); // allow interrupt
}

void timer1_init(){
	TCNT1=0x00; // ticks
	TCCR1B |= (1<<CS10); // K = 1
	OCR1A = (unsigned int) (F_CPU / 1000); // compare num
	TIMSK |= (1<<OCIE1A); // launch timer
}
```

Buzzer.h

```
#ifndef BUZZER_H
#define BUZZER_H

void buzzer_beep(int time_amount_ms);

#endif
```

Buzzer.c

```
#include <avr/io.h>
#include <util/delay.h>
```

```
void buzzer_beep(int time_amount_ms){
        PORTD |= 0x80;
        _delay_ms(time_amount_ms);
        PORTD &= 0x7F;
}
```

ПРИЛОЖЕНИЕ Б

Функциональная электрическая схема

Листов 1

ПРИЛОЖЕНИЕ В

Принципиальная электрическая схема

Листов 1

ПРИЛОЖЕНИЕ Д

Перечень радиоэлементов

Листов Х