Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

Отчёт по научно-исследовательской работе:

ТЕСТЕР ВЕДОМЫХ SPI УСТРОЙСТВ

Выполнил студент 506 группы:

_____ Д. С. Вебер

«___» _____ 2023 г.

Проверил: ст. пр. каф. ВТиЭ

_____ П. Н. Уланов

«___» _____ 2023 г.

Барнаул 2023 г.

РЕФЕРАТ

Полный объём работы составляет 20 страниц, включая * рисунков и * источника.

В ходе выполнения научно-исследовательской работы был исследован принцип работы интерфейса SPI, проведено знакомство с библиотеками для микроконтроллеров AVR и создан макет ведущего устройства.

Цель работы — создать макет тестера ведомых устройств, работающих по SPI интерфейсу.

В результате выполнения научно-исследовательской работы был создан макет управляющего устройства, с помощью которого подаются различные команды ведомому устройству, работающему по SPI интерфейсу передачи данных.

Ключевые слова: интерфейсы передачи данных, программирование микроконтроллеров.

Отчёт оформлен с помощью системы компьютерной вёрстки TEX и его расширения XETEX из дистрибутива *TeX Live*.

СОДЕРЖАНИЕ

Введение 4
1. Коротко о SPI 5
2. Создание макета устройства 7
3. Описание работы устройства 16
Заключение 17
Список использованной литературы 18

ВВЕДЕНИЕ

На сегодняшний день разрабатывается достаточно много ведомых устройств, работающих по определённым интерфейсам передачи данных. Создание каждого из них занимает время как на проектирование, так и на отладку работы.

Логично возникает вопрос с помощью чего устройства управляются и отлаживаются. Для такой задачи требуются ведущие устройства, которые передают данные ведомым. Поэтому разработка управляющих устройств несомненно остаётся необходимой и актуальной.

В данной научно-исследовательской работе объектом исследования являются интерфейсы передачи данных, программирование микроконтроллеров.

Целью работы является создать рабочий макет тестера.

Задачи научно-исследовательской работы:

1. Выбрать инструменты для разработки программы.
2. Разработать программу.
3. Собрать макет.
4. Проверить работоспособность.

1. КОРОТКО О SPI

SPI (англ. Serial Peripheral Interface, SPI bus — последовательный периферийный интерфейс, шина SPI) — последовательный синхронный стандарт передачи данных в режиме полного дуплекса, предназначенный для обеспечения простого и недорогого высокоскоростного сопряжения микроконтроллеров и периферии. Устройства, которые работают по протоколу SPI, используются в широком спектре областей, включая электронику, автомобильную промышленность, медицинское оборудование, промышленные системы и другие. Он может использоваться для передачи различных типов данных, таких как цифровые данные, команды, адреса и т.д. В зависимости от конкретной системы, которая использует протокол SPI, можно передавать следующие типы данных:

1. Цифровые данные: это наиболее распространенный тип данных, который передается по протоколу SPI. Это могут быть любые двоичные данные, например, изображения, звуковые файлы, видеофайлы, текстовые файлы и т.д.
2. Команды: некоторые периферийные устройства могут требовать отправки команд для выполнения определенных задач. Команды могут содержать информацию о том, какое действие необходимо выполнить или какой регистр необходимо изменить.
3. Адреса: если периферийное устройство имеет встроенную память, то адреса могут использоваться для чтения или записи данных в эту память.
4. Управляющие сигналы: помимо данных и адресов, протокол SPI может использоваться для передачи различных управляющих сигналов, таких как сигналы часов, выборки устройств, и т.д.

В целом, протокол SPI является очень гибким и может быть использован для передачи различных типов данных в зависимости от конкретной системы, которая использует этот протокол и существует множество устройств,

которые могут работать в качестве SPI Master. Микроконтроллеры, такие как Arduino, Raspberry Pi или STM32, имеют встроенные контроллеры SPI, которые могут быть использованы в качестве мастера. Также на рынке существуют специальные микросхемы, такие как MAX3162 и MAX3100, которые могут быть использованы для работы в качестве SPI Master. В нашем случае устройство на базе Arduino будет передавать команды, благодаря которым можно будет выяснить, что ведомое устройство работает и выполняет поставленные требования [1] [2].

2. СОЗДАНИЕ МАКЕТА УСТРОЙСТВА

Для реализации проекта была выбрана ввиду своих удобств плата на базе микроконтроллера ATmega328 — Arduino UNO со следующими характеристиками:

- Напряжение питания: 5 В.
- Цифровой ввод-вывод: 14 линий.
- Аналоговые входы: 6.
- Flash-память: 32 кб.
- Оперативная память: 2 кб.
- Встроенные интерфейсы: i2c, spi, uart, usb.

Интерфейс SPI поддерживает четыре режима работы, которые различаются настройками фазы (CPHA) и полярности (CPOL) сигнала тактирования:

1. Режим 0 (CPOL=0, CPHA=0): данные изменяются на фронте, а тактовый сигнал на спаде.
2. Режим 1 (CPOL=0, CPHA=1): данные изменяются на спаде, а тактовый сигнал на фронте.
3. Режим 2 (CPOL=1, CPHA=0): данные изменяются на спаде, а тактовый сигнал на фронте.
4. Режим 3 (CPOL=1, CPHA=1): данные изменяются на фронте, а тактовый сигнал на спаде.

В нашем устройстве интерфейс будет работать в режиме 0 на частоте 4 МГц и первым будет слаться младшим бит.

Для разработки программы была выбрана интегрированная среда разработки Arduino IDE ввиду следующих преимуществ.

1. Простота использования: интерфейс Arduino IDE довольно прост и понятен, что делает его доступным для новичков и опытных пользователей.

2. Бесплатное программное обеспечение: Arduino IDE является бесплатным программным обеспечением и может быть запущен на многих операционных системах.
3. Подробная документация: документация Arduino IDE содержит множество учебных пособий, видеоуроков и примеров проектов. Это помогает пользователям легко начать работу с этой IDE.
4. Широкое сообщество: Arduino IDE имеет очень большое количество сторонников в сообществе, которые создают новые библиотеки, расширения и советы по использованию.

В качестве устройства ввода была выбрана клавиатура, сконструированная по способу матрицирования после чего спаяна на макетной плате размером 5х7см. Кнопки были выбраны тактовые 6х6х4.3мм в количестве двадцати штук. Шестнадцать кнопок для ввода команды и четыре для функций.

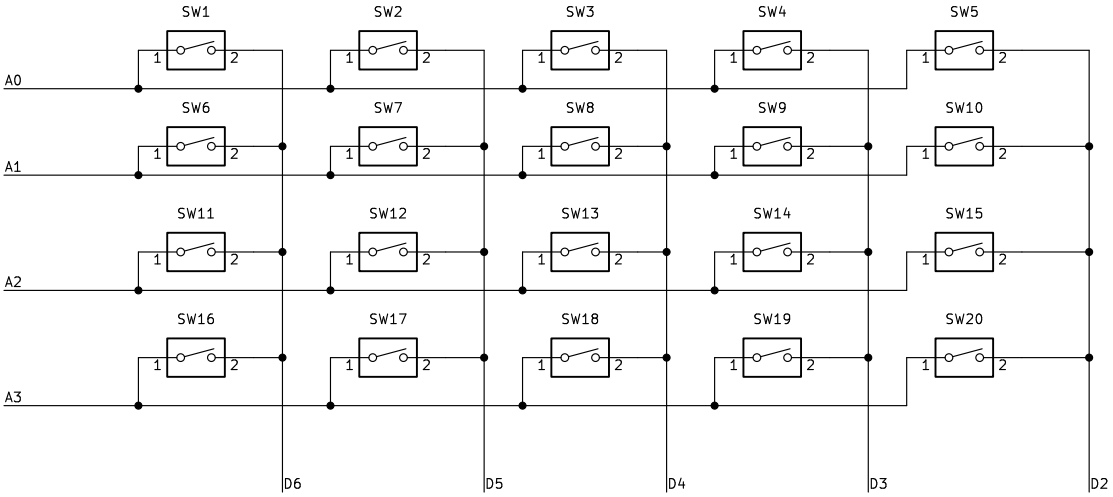

Рис. 2.1 Схема матричной клавиатуры

Для обработки клавиш потребовалась библиотека. В Arduino IDE для этих целей есть «Keypad.h». Эта библиотека предназначена для работы с матричными клавиатурами. Она позволяет считывать нажатия клавиш и определять, какая именно клавиша была нажата. После её подключения нужно определить объект класса Keypad и задать параметры его работы, такие как

количество строк и столбцов, которые есть в клавиатуре. В нашем случае клавиатура 4x5. Необходимости использовать внешние резисторы или диоды нет, так как библиотека использует внутренние подтягивающие резисторы в микроконтроллере и дополнительно обеспечивает высокое входное сопротивление на всех неиспользуемых выводах столбцов.

В листинге 2.1 мы определяем, что наша клавиатура содержит 4 строки и 5 столбцов, а также задаем, какие символы соответствуют каким клавишам. Затем мы указываем, к каким пинам на Arduino подключены строки и столбцы клавиатуры. После этого, используя метод customKeypad.getKey() считываем нажатие клавиш.

Листинг 2.1

Инициализация матричной клавиатуры

```
#include <Keypad.h>
const byte ROWS = 4;
const byte COLS = 5;
char button;
char keys[ROWS][COLS] = {  // раскладка клавиатуры
  { '0', '1', '2', '3', 'h' },
  { '4', '5', '6', '7', 'x' },
  { '8', '9', 'A', 'B', 's' },
  { 'C', 'D', 'E', 'F', 'd' }
};
byte rowPins[ROWS] = { A0, A1, A2, A3 };  // подключение к строкам клавиатуры
byte colPins[COLS] = { 6, 5, 4, 3, 2 };   // подключение к столбцам клавиатуры
Keypad customKeypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS); // инициализация
↪  клавиатуры

void loop() {
  button = customKeypad.getKey();  // определение нажатой кнопки
}
```

Для обработки кнопок была написана функция detectbuttons (листинг 2.2). На вход подпрограммы ничего не поступает и никаких значений она не возвращает. Функция вызывает нужную подпрограмму по нажатой клавише. Рисунок 2.2 поясняет ход работы подпрограммы.

Листинг 2.2

Функция обработки кнопок

```
void detectbuttons() {
  switch (button) {
    case '0':
      //    Serial.println("Button 0");
```

```
5         append_cmd(0x0000);
6         break;
7
8       ...
9
10      case 'F':
11        //    Serial.println("Button F");
12        append_cmd(0x000F);
13        break;
14
15      case 's':
16        //    Serial.println("Button send");
17        pack[0] = cmd;
18        pack[1] = cmd >> 8;
19        digitalWrite(SS, LOW);
20        for (int i = 0; i < 2; i++)
21          SPI.transfer(pack[i]);
22        digitalWrite(SS, HIGH);
23        delay(1000);
24        break;
25
26      case 'd':
27        //    Serial.println("Button del");
28        cmd = cmd >> 4;
29        break;
30
31      case 'h':  //help
32        do {
33          myOLED.clrScr();
34          myOLED.print("Help", CENTER, 0);
35          myOLED.print("0 1 2 3 h", CENTER, 16);
36          myOLED.print("4 5 6 7 x", CENTER, 26);
37          myOLED.print("8 9 A B s", CENTER, 36);
38          myOLED.print("C D E F d", CENTER, 46);
39          myOLED.update();
40          button = customKeypad.getKey();
41          detectbuttons();
42        } while (!button);
43        cmd = 0x0000;
44        break;
45    }
46  }
```

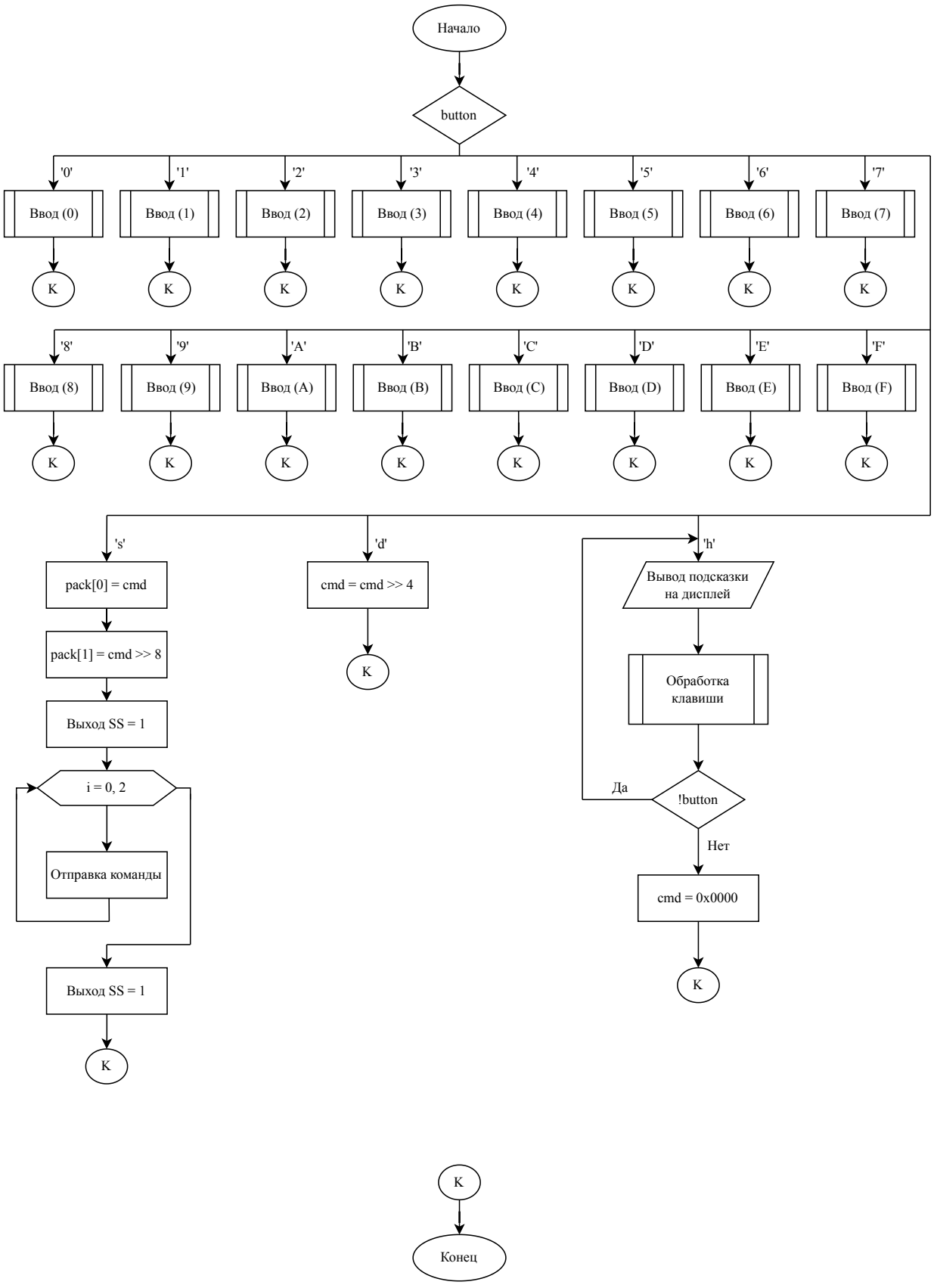

Рис. 2.2 Блок-схема подпрограммы обработки кнопок.

Для ввода команды была написала функция append_cmd (листинг 2.3). Подпрограмма только принимает. На вход поступает, вводимый пользователем бит, а далее записывается в команду. Ход работы поясняется на рисунке 2.3.

Листинг 2.3

Функция ввода команды

```
void append_cmd(uint16_t comand) {
  if (cmd == 0) {
    cmd = comand;
  } else {
    cmd = cmd << 4;
    cmd |= comand;
  }
}
```

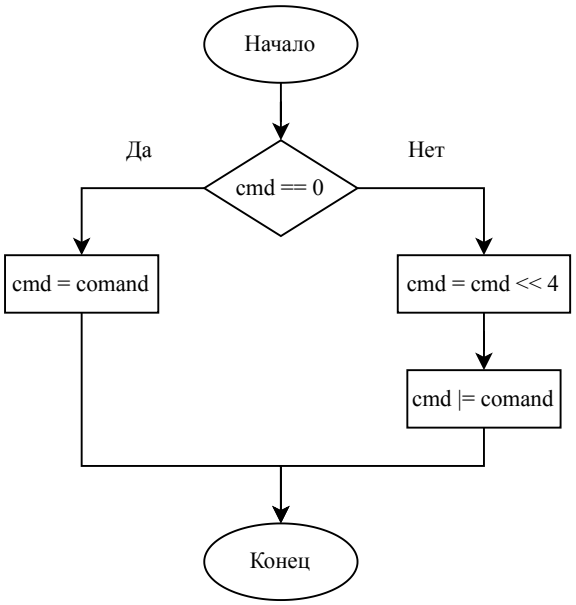

Рис. 2.3 Блок-схема подпрограммы ввода команды.

В качестве устройства вывода был выбран OLED дисплей. У данного индикатора пиксели излучают свет сами, изображение получается более контрастным и насыщенным, с хорошим углом обзора. Ко всему этому у индикатора низкое энергопотребление. Имеет разрешение 128 на 64 точек, управляется по интерфейсу I2C, графический чип SSD1306 и питается от 3 — 5 В. Для работы с ним была выбрана библиотека «OLED_I2C.h» за её простоту и

лёгкость. Сам же дисплей нужен для отображения команды, которую вводит пользователь.

Листинг 2.4

Вывод команды на дисплей

```
#include <OLED_I2C.h>
OLED myOLED(SDA, SCL);
extern uint8_t SmallFont[];
uint16_t cmd;
void setup() {
myOLED.begin();
myOLED.setFont(SmallFont);
myOLED.clrScr();
myOLED.update();
}

void loop() {
myOLED.clrScr();
myOLED.print(String(cmd, HEX), CENTER, 25);  //выводим на экран вводимую команду
myOLED.update();
}
```

Полный код программы приведён в приложении. После успешной компиляции среда разработки вывело сообщение о том, что программа использует 6892 байт (21%) памяти устройства. Всего доступно 32256 байт.

Блок-схема на рисунке 2.4 поясняет работу программы.

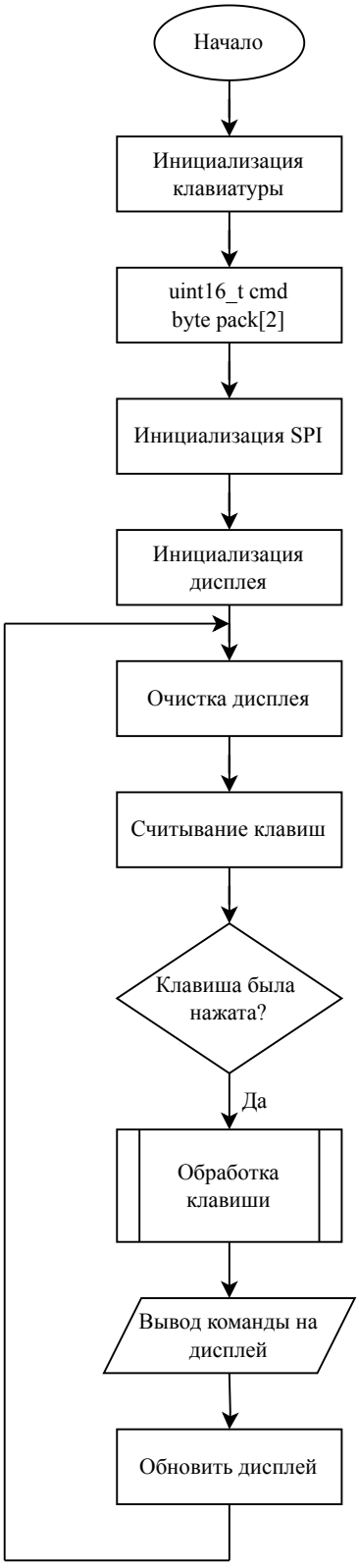

Рис. 2.4 Блок-схема главной программы.

Макет описывается следующей функциональной схемой (рис. 2.5). Подключение клавиатуры может быть осуществлено к любым свободным пинам. В нашем случае строки подключаются к аналоговым пинам 0, 1, 2, 3, а столбцы к цифровым 6, 5, 4, 3, 2. Дисплей подключен по шине I2C, а линии SPI идут к ведомому устройству.

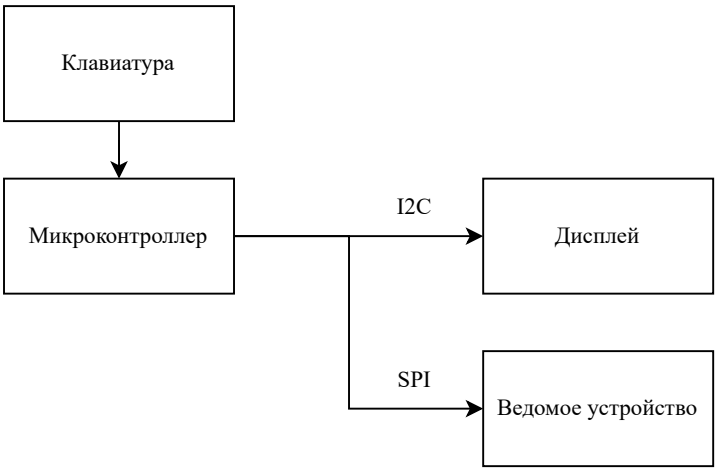

Рис. 2.5 Функциональная схема SPI тестера.

3. ОПИСАНИЕ РАБОТЫ УСТРОЙСТВА

Процесс работы пользователя с устройством можно представить следующей структурной схемой (рис.4.1).

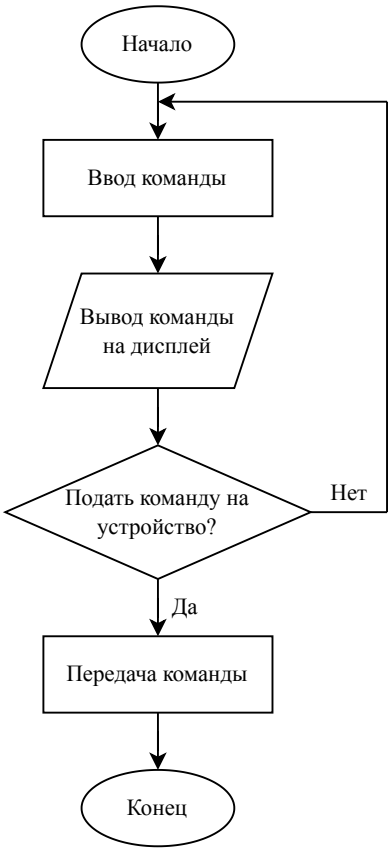

Рис. 3.1 Структурная схема.

ЗАКЛЮЧЕНИЕ

В результате научно-исследовательской работы были выполнены следующие задачи:

1. Выбраны инструменты для разработки программы.
2. Разработана программа.
3. Собран макет.
4. Проверена работоспособность.

По итогам работы цель достигнута: создан рабочий макет тестера ведомых SPI устройств.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. *Felker D.* Android Application Development For Dummies. — Wiley, 2010. — ISBN 9781118005156. — URL: https://books.google.ru/books?id=1C3yNgqZnUkC.
2. Github [Электронный ресурс] Википедия — свободная энцциклопедия. Режим доступа: https://ru.wikipedia.org/wiki/GitHub. — (Дата обр. 18.06.2023).

ПРИЛОЖЕНИЕ

Текст программы

Тестер ведомых SPI устройств

```
#include <SPI.h>
#include <OLED_I2C.h>
OLED myOLED(SDA, SCL);
extern uint8_t SmallFont[];
#include <Keypad.h>
const byte ROWS = 4;
const byte COLS = 5;
char button;
char keys[ROWS][COLS] = {
  { '0', '1', '2', '3', 'h' },
  { '4', '5', '6', '7', 'x' },
  { '8', '9', 'A', 'B', 's' },
  { 'C', 'D', 'E', 'F', 'd' }
};
byte rowPins[ROWS] = { A0, A1, A2, A3 };  // подключение к строкам клавиатуры
byte colPins[COLS] = { 6, 5, 4, 3, 2 };   // подключение к столбцам клавиатуры
Keypad customKeypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
uint16_t cmd;
byte pack[2];
void setup() {
  //Serial.begin(9600);
  SPI.begin();
  myOLED.begin();
  myOLED.setFont(SmallFont);
  myOLED.clrScr();
  myOLED.update();
}

void loop() {
  myOLED.clrScr();
  button = customKeypad.getKey();  // определение нажатой кнопки
  if (button != NO_KEY)
    detectbuttons();
  //Serial.println(button);
  myOLED.print(String(cmd, HEX), CENTER, 25);  //выводим на экран вводимую команду
  myOLED.update();
}


void append_cmd(uint16_t comand) {
  if (cmd == 0) {
    cmd = comand;
  } else {
    cmd = cmd << 4;
    cmd |= comand;
  }
}

void detectbuttons() {
  switch (button) {
    case '0':
      //    Serial.println("Button 0");
      append_cmd(0x0000);
      break;
```

```
55
56      case '1':
57        //    Serial.println("Button 1");
58        append_cmd(0x0001);
59        break;
60
61      case '2':
62        //    Serial.println("Button 2");
63        append_cmd(0x0002);
64        break;
65
66      case '3':
67        //    Serial.println("Button 3");
68        append_cmd(0x0003);
69        break;
70
71      case '4':
72        //    Serial.println("Button 4");
73        append_cmd(0x0004);
74        break;
75
76      case '5':
77        //    Serial.println("Button 5");
78        append_cmd(0x0005);
79        break;
80
81      case '6':
82        //    Serial.println("Button 6");
83        append_cmd(0x0006);
84        break;
85
86      case '7':
87        //    Serial.println("Button 7");
88        append_cmd(0x0007);
89        break;
90
91      case '8':
92        //    Serial.println("Button 8");
93        append_cmd(0x0008);
94        break;
95
96      case '9':
97        //    Serial.println("Button 9");
98        append_cmd(0x0009);
99        break;
100
101     case 'A':
102       //    Serial.println("Button A");
103       append_cmd(0x000A);
104       break;
105
106     case 'B':
107       //    Serial.println("Button B");
108       append_cmd(0x000B);
109       break;
110
111     case 'C':
112       //    Serial.println("Button C");
113       append_cmd(0x000C);
114       break;
115
116     case 'D':
117       //    Serial.println("Button D");
118       append_cmd(0x000D);
```

```
119       break;
120
121     case 'E':
122       //    Serial.println("Button E");
123       append_cmd(0x000E);
124       break;
125
126     case 'F':
127       //    Serial.println("Button F");
128       append_cmd(0x000F);
129       break;
130
131     case 's':
132       //    Serial.println("Button send");
133       pack[0] = cmd;
134       pack[1] = cmd >> 8;
135       digitalWrite(SS, LOW);
136       for (int i = 0; i < 2; i++)
137         SPI.transfer(pack[i]);
138       digitalWrite(SS, HIGH);
139       delay(1000);
140       break;
141
142     case 'd':
143       //    Serial.println("Button del");
144       cmd = cmd >> 4;
145       break;
146
147     case 'h':  //help
148       do {
149         myOLED.clrScr();
150         myOLED.print("Help", CENTER, 0);
151         myOLED.print("0 1 2 3 h", CENTER, 16);
152         myOLED.print("4 5 6 7 x", CENTER, 26);
153         myOLED.print("8 9 A B s", CENTER, 36);
154         myOLED.print("C D E F d", CENTER, 46);
155         myOLED.update();
156         button = customKeypad.getKey();
157         detectbuttons();
158       } while (!button);
159       cmd = 0x0000;
160       break;
161   }
162 }
163
```