

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 22 страницы, включая 8 рисунков.

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 оформлена с помощью системы компьютерной вёрстки $\text{T}_{\text{E}}\text{X}$ и его расширения $\text{X}_{\text{E}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ из дистрибутива *TeX Live*.

СОДЕРЖАНИЕ

Введение	4
1. Коротко о SPI	5
2. Создание макета устройства	7
3. Описание работы устройства	17
Заключение	20
Список использованной литературы	21

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 [1]. 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. В нашем случае устройство на базе Arduino будет передавать команды, благодаря которым можно будет выяснить, что ведомое устройство работает и выполняет поставленные требования.

2. СОЗДАНИЕ МАКЕТА УСТРОЙСТВА

Для реализации проекта была выбрана ввиду своих удобств плата на базе микроконтроллера ATmega328 — Arduino UNO со следующими характеристиками[2]:

- Напряжение питания: 5 В.
- Цифровые входы/выходы: 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]:

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 [3].

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.

Для работы с интерфейсом SPI была выбрана штатная библиотека «SPI.h» [4]. Команда SPI.begin() инициализирует шину SPI, устанавливая пины SCK, MOSI и SS на выходы, MISO на вход. Выход SS по умолчанию в высоком уровне. Передача байта осуществляется с помощью команды SPI.transfer().

Листинг 2.1

Передача команды 0xABCD по SPI

```

1 #include <SPI.h>
2 uint16_t cmd = 0xabcd;
3 byte pack[2];
4 void setup() {
5     SPI.begin();
6 }
7
8 void loop() {
9     pack[0] = cmd;
10    pack[1] = cmd >> 8;
11    digitalWrite(SS, LOW);
12    for (int i = 0; i < 2; i++)
13        SPI.transfer(pack[i]);
14    digitalWrite(SS, HIGH);
15 }

```

В качестве устройства ввода была выбрана клавиатура, сконструированная по способу матрицирования после чего спаяна на макетной плате размером 5x7см. Кнопки были выбраны тактовые 6x6x4.3мм в количестве двадцати штук. Шестнадцать кнопок для ввода команды и четыре для функций.

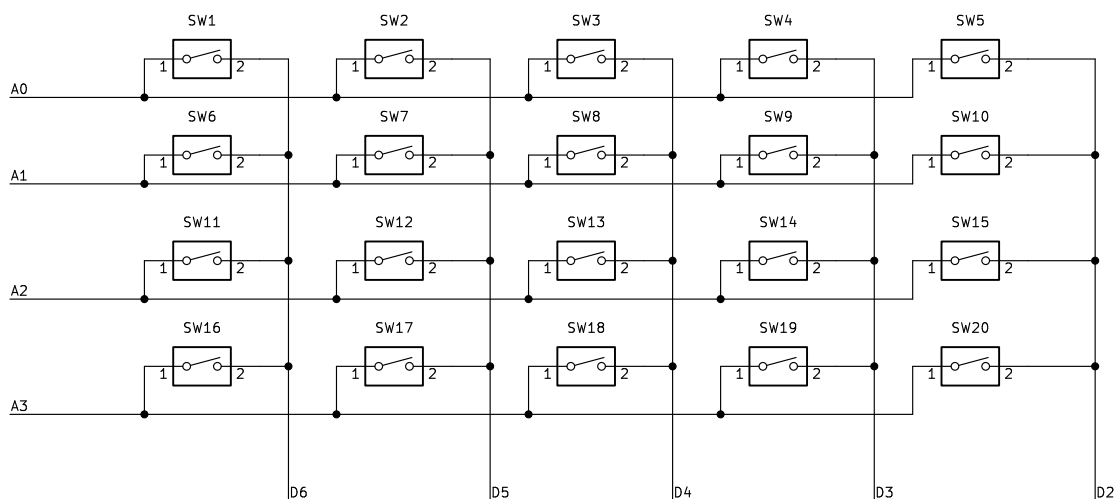


Рис. 2.1 Схема матричной клавиатуры.

Для обработки клавиш потребовалась библиотека. В Arduino IDE для этих целей есть «Keypad.h» [5]. 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. Нecessности 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 2.2 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 2.2

Инициализация матричной клавиатуры

```

1 #include <Keypad.h>
2 const byte ROWS = 4;
3 const byte COLS = 5;
4 char button;
5 char keys[ROWS][COLS] = { // раскладка клавиатуры
6   { '0', '1', '2', '3', 'h' },
7   { '4', '5', '6', '7', 'x' },
8   { '8', '9', 'A', 'B', 's' },
9   { 'C', 'D', 'E', 'F', 'd' }
10 };
11 byte rowPins[ROWS] = { A0, A1, A2, A3 }; // подключение к строкам клавиатуры
12 byte colPins[COLS] = { 6, 5, 4, 3, 2 }; // подключение к столбцам клавиатуры
13 Keypad customKeypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS); // инициализация клавиатуры
14
15 void loop() {
16   button = customKeypad.getKey(); // определение нажатой кнопки
17 }

```

Для обработки кнопок была написана функция `detectbuttons` (листинг 2.3). 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 2.3

Функция обработки кнопок

```

1 void detectbuttons() {
2   switch (button) {
3     case '0':
4       // 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 }
```

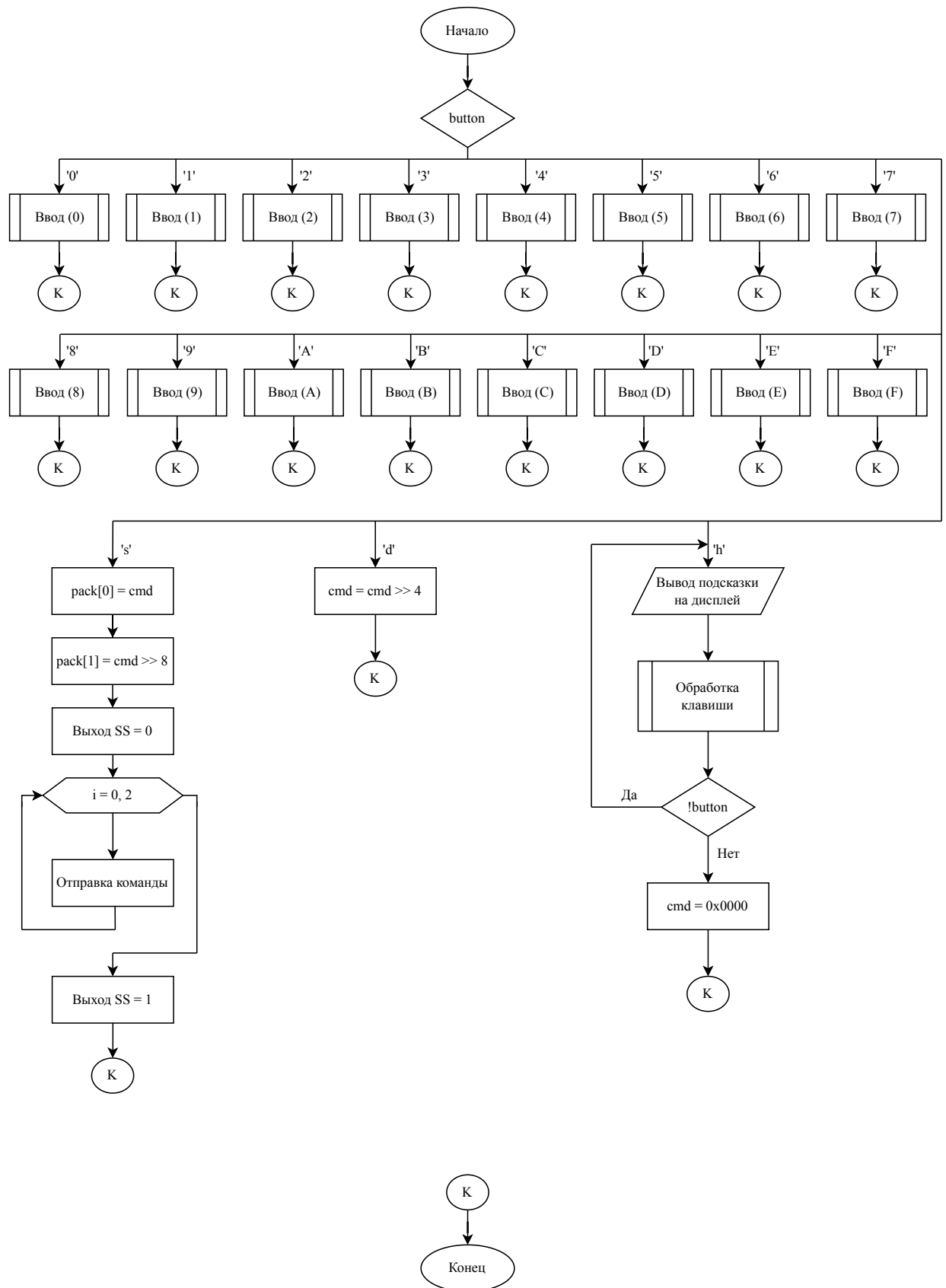


Рис. 2.2 Блок-схема подпрограммы обработки кнопок.

Для ввода команды была написана функция `append_cmd` (листинг 2.4). Подпрограмма только принимает. На вход поступает, вводимый пользователем набор бит, а далее записывается в команду. Ход работы поясняется на рисунке 2.3.

Листинг 2.4

Функция ввода команды

```

1 void append_cmd(uint16_t comand) {
2   if (cmd == 0) {
3     cmd = comand;
4   } else {
5     cmd = cmd << 4;
6     cmd |= comand;
7   }
8 }

```

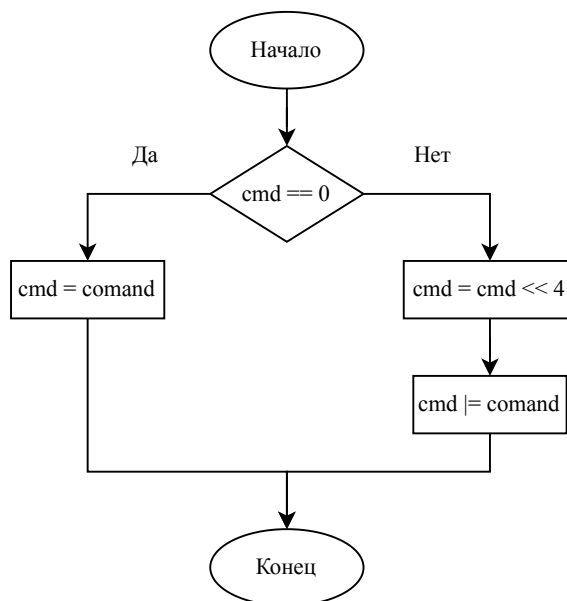


Рис. 2.3 Блок-схема подпрограммы ввода команды.

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 [6]. Сам же дисплей нужен для отображения команды, которую вводит пользователь.

Листинг 2.5

Вывод команды на дисплей

```
1 #include <OLEDD_I2C.h>
2 OLED myOLED(SDA, SCL);
3 extern uint8_t SmallFont[];
4 uint16_t cmd;
5 void setup() {
6   myOLED.begin();
7   myOLED.setFont(SmallFont);
8   myOLED.clrScr();
9   myOLED.update();
10 }
11
12 void loop() {
13   myOLED.clrScr();
14   myOLED.print(String(cmd, HEX), CENTER, 25); //выводим на экран вводимую команду
15   myOLED.update();
16 }
```

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.

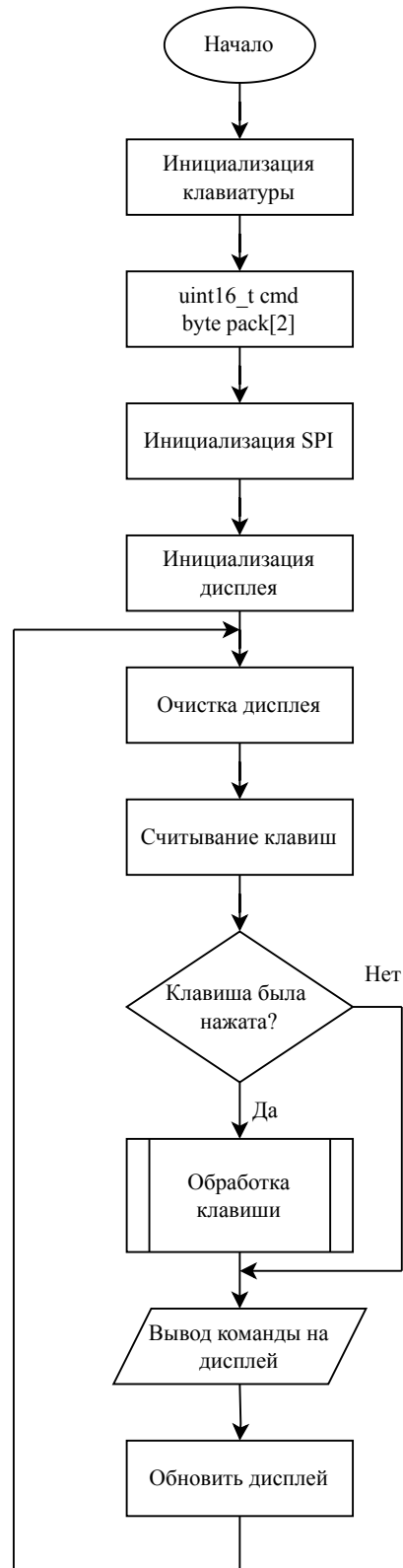


Рис. 2.4 Блок-схема главной программы.

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.

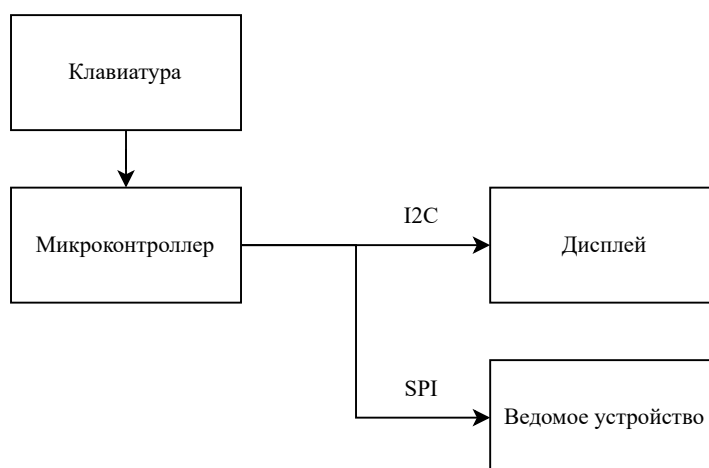


Рис. 2.5 Функциональная схема SPI тестера.

3. ОПИСАНИЕ РАБОТЫ УСТРОЙСТВА

Процесс работы пользователя с устройством можно представить следующей структурной схемой (рис.3.1).

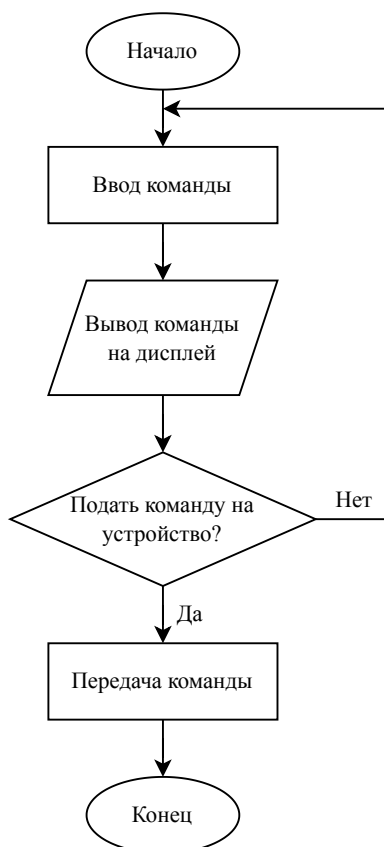


Рис. 3.1 Структурная схема работы устройства.

Как упоминалось ранее устройством ввода является матричная клавиатура. Кроме клавиш для ввода команды на ней расположены клавиши вызова подсказки с раскладкой клавиатуры «h», обратный пробел «d» и кнопка отправки команды «s». Клавиша «x» не назначена и может в будущем быть использована. Всё рабочее время устройство ожидает ввода команды и выводит текущее значение на дисплей. Когда пользователь ввёл команду и хочет её отправить он нажимает кнопку «s». После чего формируется посылка из двух пакетов по одному байту, линия выбора ведомого переходит в низкий уровень и начинается отправка команды. По окончании отправки линия переходит обратно в высокий уровень.

В качестве ведомого устройства для тестов был выбран портативный четырехканальный логический анализатор Miniware MiniDSO LA104. На рисунке 3.2 изображены результаты отправки команд с тестера.

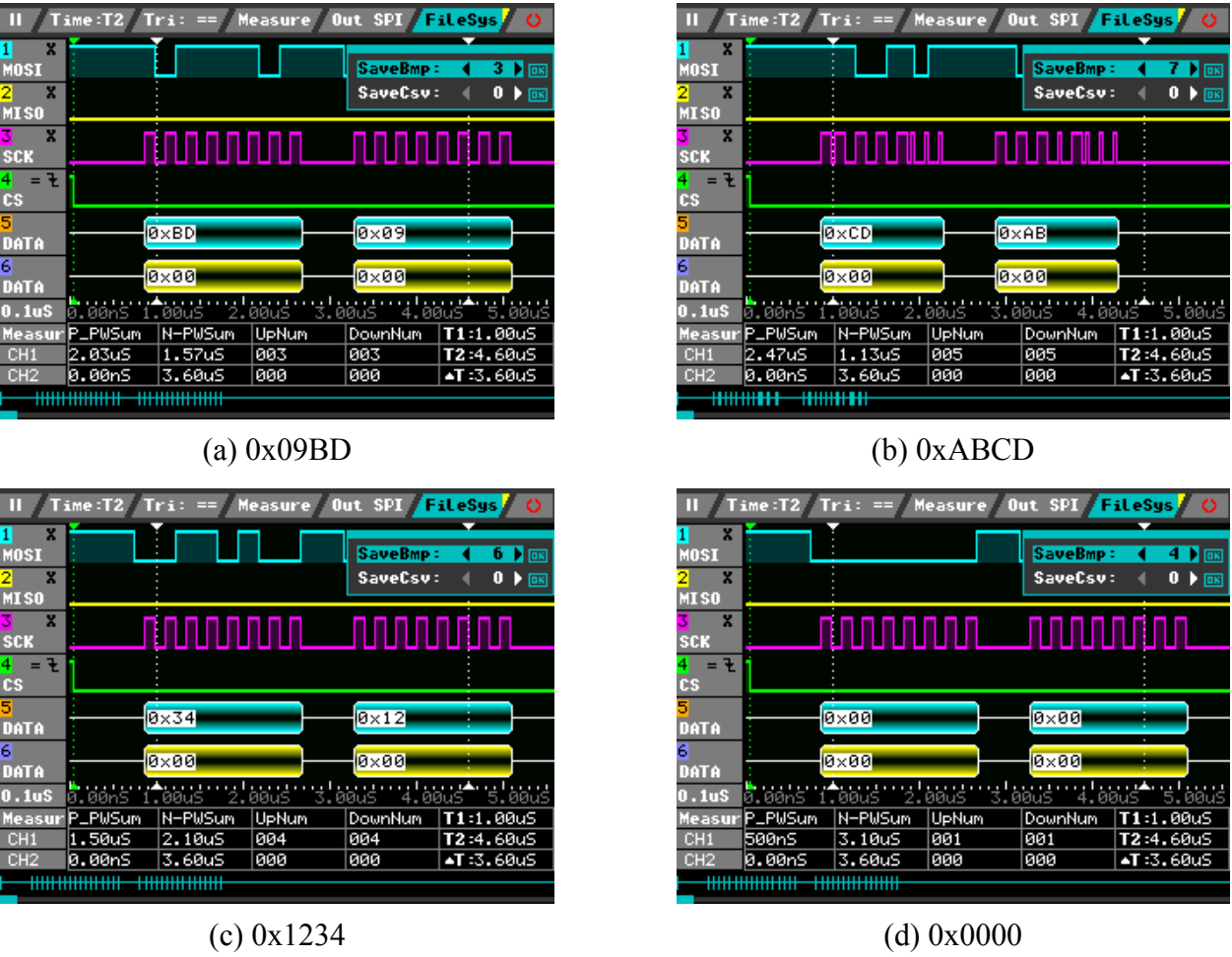


Рис. 3.2 Данные с анализатора.

Проверим нашу заявленную частоту синхронизации 4 МГц (рис. 3.3).

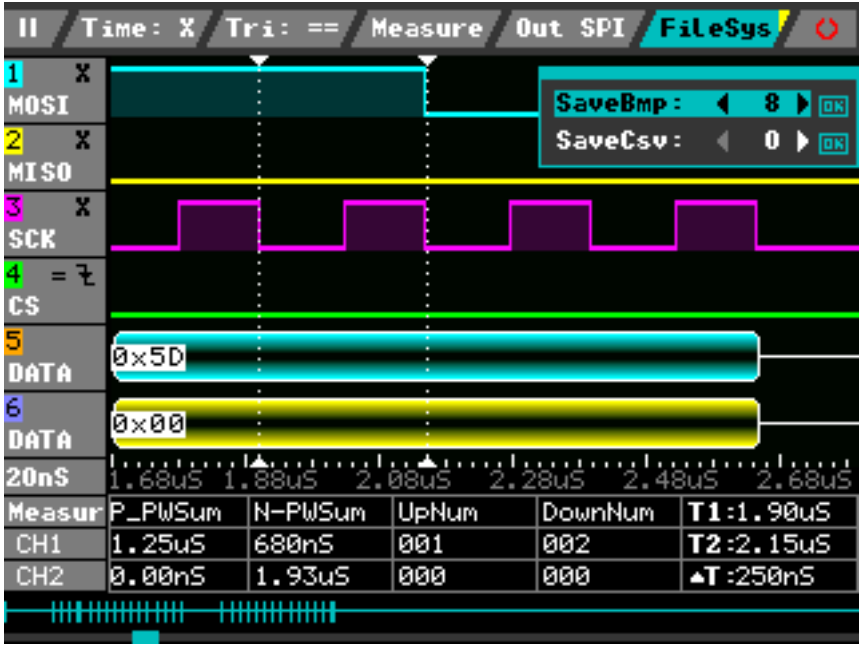


Рис. 3.3 Измерение частоты тактового сигнала.

Установим флаги $T1$ и $T2$ на начало и конец периода синхросигнала и узнаем его длину, а затем вычислим частоту.

$$\Delta T = T2 - T1 = 250 \text{ нс.}$$

$$f = \frac{1}{\Delta T} = \frac{1}{250 \cdot 10^{-9}} = 4 \text{ МГц.}$$

Таким образом, макет тестера ведомых SPI устройств является работоспособным. Частота и отсылаемые команды были подтверждены с помощью логического анализатора.

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. Serial Peripheral Interface [Электронный ресурс] Википедия — свободная энциклопедия. Режим доступа: https://ru.wikipedia.org/wiki/Serial_Peripheral_Interface. — (Дата обр. 14.02.2023).
2. Даташит Arduino UNO [Электронный ресурс] Официальный сайт компании Arduino. Режим доступа: <https://docs.arduino.cc/hardware/uno-rev3>. — (Дата обр. 02.03.2023).
3. Среда разработки Arduino IDE [Электронный ресурс] Официальный сайт компании Arduino. Режим доступа: <https://www.arduino.cc/en/software>. — (Дата обр. 03.03.2023).
4. Библиотека SPI [Электронный ресурс] Официальный сайт компании Arduino. Режим доступа: <https://www.arduino.cc/reference/en/language/functions/communication/spi/>. — (Дата обр. 15.03.2023).
5. Библиотека Keypad [Электронный ресурс] Официальный сайт компании Arduino. Режим доступа: <https://playground.arduino.cc/Code/Keypad/>. — (Дата обр. 23.03.2023).
6. Библиотека OLED_I2C [Электронный ресурс] GitHub. Режим доступа: https://github.com/jlegas/OLED_I2C. — (Дата обр. 31.03.2023).

ПРИЛОЖЕНИЕ

Текст программы

Тестер ведомых SPI устройств

```

1 #include <SPI.h>
2 #include <OLED_I2C.h>
3 OLED myOLED(SDA, SCL);
4 extern uint8_t SmallFont[];
5 #include <Keypad.h>
6 const byte ROWS = 4;
7 const byte COLS = 5;
8 char button;
9 char keys[ROWS][COLS] = {
10   { '0', '1', '2', '3', 'h' },
11   { '4', '5', '6', '7', 'x' },
12   { '8', '9', 'A', 'B', 's' },
13   { 'C', 'D', 'E', 'F', 'd' }
14 };
15 byte rowPins[ROWS] = { A0, A1, A2, A3 }; // подключение к строкам клавиатуры
16 byte colPins[COLS] = { 6, 5, 4, 3, 2 }; // подключение к столбцам клавиатуры
17 Keypad customKeypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
18 uint16_t cmd;
19 byte pack[2];
20 void setup() {
21   //Serial.begin(9600);
22   SPI.begin();
23   myOLED.begin();
24   myOLED.setFont(SmallFont);
25   myOLED.clrScr();
26   myOLED.update();
27 }
28
29 void loop() {
30   myOLED.clrScr();
31   button = customKeypad.getKey(); // определение нажатой кнопки
32   if (button != NO_KEY)
33     detectbuttons();
34   //Serial.println(button);
35   myOLED.print(String(cmd, HEX), CENTER, 25); //выводим на экран вводимую команду
36   myOLED.update();
37 }
38
39
40 void append_cmd(uint16_t comand) {
41   if (cmd == 0) {
42     cmd = comand;
43   } else {
44     cmd = cmd << 4;
45     cmd |= comand;
46   }
47 }
48
49 void detectbuttons() {
50   switch (button) {
51     case '0':
52       // Serial.println("Button 0");
53       append_cmd(0x0000);
54       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         break;
140
141     case 'd':
142         // Serial.println("Button del");
143         cmd = cmd >> 4;
144         break;
145
146     case 'h': //help
147         do {
148             myOLED.clrScr();
149             myOLED.print("Help", CENTER, 0);
150             myOLED.print("0 1 2 3 h", CENTER, 16);
151             myOLED.print("4 5 6 7 x", CENTER, 26);
152             myOLED.print("8 9 A B s", CENTER, 36);
153             myOLED.print("C D E F d", CENTER, 46);
154             myOLED.update();
155             button = customKeypad.getKey();
156             detectbuttons();
157         } while (!button);
158         cmd = 0x0000;
159         break;
160     }
161 }
162

```
