Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ ИМЕНИ ПЕРВОГО ПРЕЗИДЕНТА РОССИИ Б.Н. ЕЛЬЦИНА

(УрФУ)

Физико-технологический институт

Кафедра экспериментальной физики

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члены комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**АВТОМАТИЗАЦИЯ МОНОХРОМАТОРА**

КУРСОВОЙ ПРОЕКТ

по дисциплине «Проектирование электронных приборов»

Пояснительная записка

14.05.04 27.20.23.130 ПЗ

|  |  |  |
| --- | --- | --- |
| Руководитель  профессор, д.ф.-м.н. | \_\_\_\_\_\_\_\_\_\_\_\_\_ | А.В. Ищенко |
|  |  |  |
| Нормоконтролер  вед. инж. | \_\_\_\_\_\_\_\_\_\_\_\_\_ | Ю.Г. Лазарев |
|  |  |  |
| Студент гр. ФТ-570301 | \_\_\_\_\_\_\_\_\_\_\_\_\_ | Д.С. Егоров |

Екатеринбург

2022

Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное

Учреждение высшего профессионального образования

**«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»**

«УТВЕРЖДАЮ»

Зав.кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

**Задание № \_\_\_\_\_**

**на курсовой проект/работу**

Студент группы \_\_\_\_Фт-570301\_\_\_ специальность/направление\_\_\_Электроника и автоматика физических установок\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Фамилия \_\_\_Егоров\_\_\_\_\_\_\_ Имя \_\_\_\_\_Дмитрий\_\_\_\_\_\_\_ Отчество \_\_Сергеевич\_\_\_\_\_\_\_\_

Руководитель курсового проекта/работы \_\_\_Ищенко Алексей Владимирович\_\_\_\_\_\_\_\_\_\_\_

Срок проектирования с \_­­­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_01.09.21\_\_\_\_\_\_\_\_\_ по \_\_\_\_\_\_\_\_\_\_29.12.21\_\_\_\_\_\_\_\_\_

1. Тема курсового проекта/работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Автоматизация монохроматора\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Содержание проекта Проектирование и расчет параметров требуемого устройства

3. Особые дополнительные сведения \_

4. План выполнения курсового проекта/работы

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование элементов проектной работы | Сроки | Примечание | Отметка о выполнении |
| Получение задания | 01.09.21 – 14.09.21 |  |  |
| Разработка структурной схемы | 15.09.21 – 15.10.21 |  |  |
| Исследование и выбор аппаратной реализации | 16.10.21 – 16.11.21 |  |  |
| Расчет индикатора заряда | 16.11.21-08.12.21 |  |  |
| Исследование возможности снятия заряда МК | 09.12.21-16.12.21 |  |  |
| Подготовка ПЗ и сдача его на проверку | 17.12.21-29.12.21 |  |  |
| Защита | 28.01.22 |  |  |

5. Курсовой проект/работа закончен \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Оценка проекта/работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 6](#_Toc93962734)

[ВВЕДЕНИЕ 7](#_Toc93962734)

[1 Устройство системы 8](#_Toc93962735)

[1.1 Блок для работы с шаговым двигателем 9](#_Toc93962736)

[1.2 Блок для работы с ФЭУ. 11](#_Toc93962737)

[1.3 Взаимодействие с сервером. 12](#_Toc93962738)

[1.4 Питание схемы. 13](#_Toc93962739)

[1.5 Вывод по разделу 13](#_Toc93962740)

[2 Схемотехническая реализация 14](#_Toc93962741)

[2.1 STM32……… 14](#_Toc93962742)

[2.2 ESP8266-01 17](#_Toc93962743)

[2.3 SDM660…. 18](#_Toc93962744)

[2.4 Блок для получения сигнала с ФЭУ. 19](#_Toc93962745)

[3 Кодовая база 22](#_Toc93962746)

[3.1 .NET сервер 22](#_Toc93962747)

[Код на STM32. 26](#_Toc93962748)

[Код на ESP–01 29](#_Toc93962749)

[Заключение 29](#_Toc93962750)

[ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА 30](#_Toc93962751)

[ПРИЛОЖЕНИЕ А 32](#_Toc93962752)

[ПРИЛОЖЕНИЕ Б 34](#_Toc93962753)

[ПРИЛОЖЕНИЕ В 36](#_Toc93962754)

[ПРИЛОЖЕНИЕ Г 40](#_Toc93962755)

[ПРИЛОЖЕНИЕ Д 46](#_Toc93962756)

[ПРИЛОЖЕНИЕ Е 49](#_Toc93962757)

[ПРИЛОЖЕНИЕ Ж 50](#_Toc93962759)

[ПРИЛОЖЕНИЕ З 51](#_Toc93962761)

[ПРИЛОЖЕНИЕ И 52](#_Toc93962763)

[ПРИЛОЖЕНИЕ Й 53](#_Toc93962765)

[ПРИЛОЖЕНИЕ К 54](#_Toc93962767)

[ПРИЛОЖЕНИЕ Л 57](#_Toc93962768)

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

STM32 – 32 битный микроконтроллер

ESP8266 – микроконтроллер с интерфейсом Wi-Fi

.NET – платформа для разработки веб-приложений

ФЭУ – фотоэлектронный умножитель

24CS22C-400 – шаговый двигатель

SDM660– драйвер шагового двигателя

LMC7101– операционный усилитель

MQTT – протокол передачи данных

МК – микроконтроллер

UART – интерфейс взаимодействия

ОУ – операционный усилитель

WEB – всемирная сетевая паутина

C# – язык программирования

F# – язык программирования

Visual Basic – язык программирования

CLI – интерфейс командной строки

SDK – инструмент для компиляции кода

MSBuild – система сборок

NuGet – менеджер пакетов

HTTP – протокол передачи данных

MVC **–** архитектурный паттерн для кода

**ВВЕДЕНИЕ**

Благодаря широкому распространению беспроводных технологий автоматизация управления электронными приборами получила большую популярность. Появилось множество библиотек и модулей прикладного уровня для интеграции в существующие устройства, не имеющих доступа к сети. Теперь разработчику не требуется проектировать взаимодействие на всех уровнях OSI, а достаточно оперировать прикладным слоем.

Целью данной работы является разработка и интеграция системы дистанционного управления монохроматором. В работе монохроматора возможна автоматизация таких процессов как: поворот и считывание состояния диспергирующего элемента, считывание состояния ФЭУ, передача и агрегация этих данных на компьютере без ручного переноса значений.

Сама система из STM32, ESP8266 и .NET сервера является универсальной с точки зрения своей архитектуры для подключения устройства к сети, что позволяет решать большой спектр различных задач.

1. **Устройство системы**

Целью данного раздела является:

1. Рассмотреть принцип работы каждой части системы;
2. Составить полную картину функционирования устройства со связью логических блоков.

На рисунке 1 изображена структурная схема разработанного устройства.

Микро-контроллер STM32

Wi-Fi–модуль ESP8266–01

NET  
сервер

Веб-интерфейс

Пользователь

Монохроматор

ФЭУ

База данных

Рисунок 1 – Структурная схема устройства.

Логические блоки устройства:

1. Блок для управления монохроматором;
2. Блок для работы с ФЭУ;
3. Блок для передачи данных на сервер;
4. Сервер для сбора и вывода данных.
   1. **Блок для работы с шаговым двигателем**

Принцип работы монохроматора основан на выделении волны определенной частоты из сплошного спектра путем изменения угла поворота диспергирующего элемента, состоящего из призмы и дифракционной решетки. Для более точной работы поворот диспергирующего элемента осуществляется специальным передаточным механизмом.

Для автоматизации вращения будем использовать двухфазный шаговый двигатель 24CS22C-400 [Рисунок 2]. Одной из основных характеристик шагового двигателя является угловое перемещение. Величина данного параметра задает количество шагов на полный оборот двигателя. В данной реализации было установлено, что шаг в 1,8 градуса (200 шагов/оборот) является достаточным.

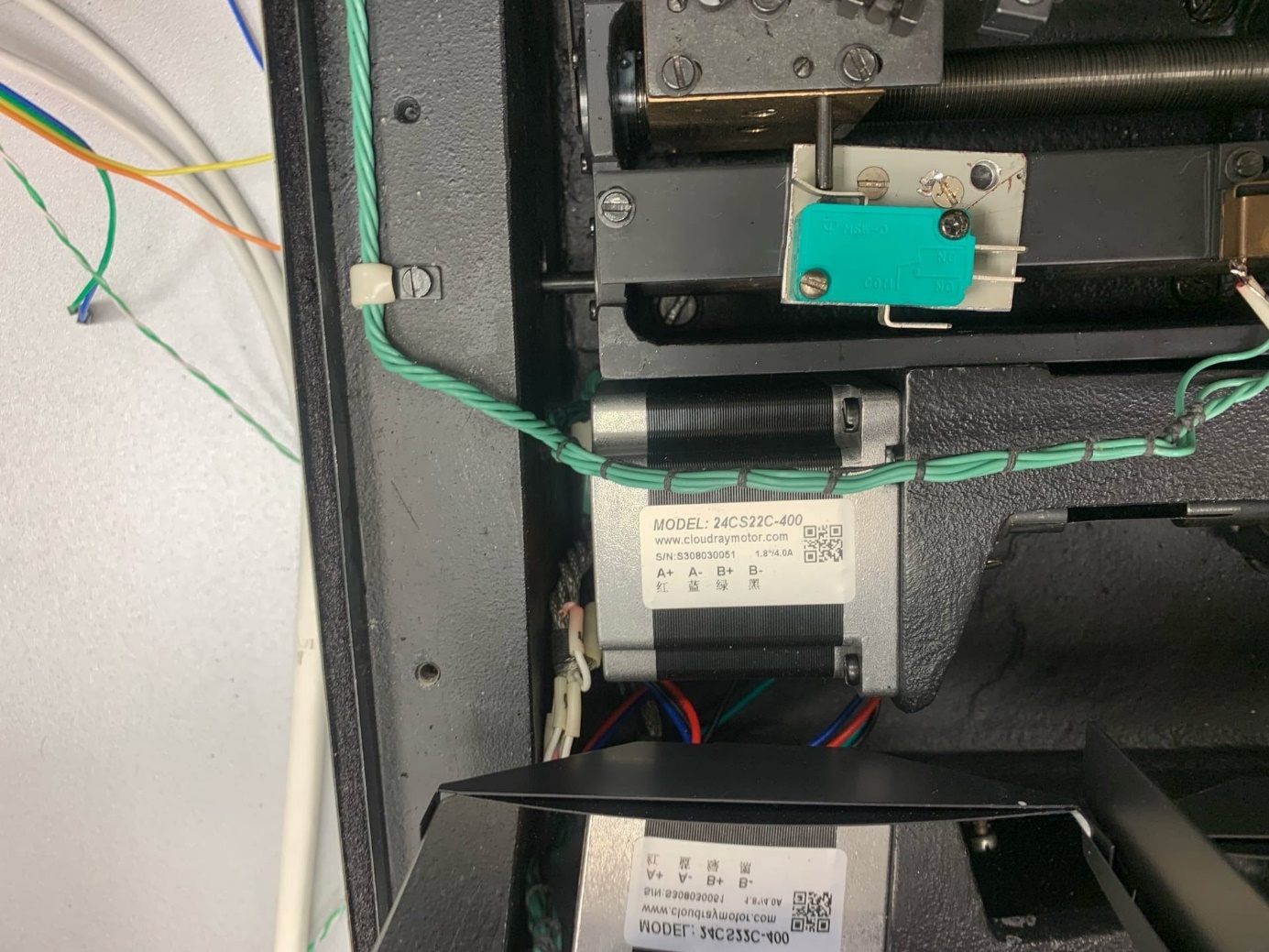


Рисунок 2 – Шаговый двигатель 24CS22C-400.

Для управления шаговым двигателем будем использовать специализированный драйвер SDM660 [Рисунок 3]. Управление двигателем обеспечивается за счет изменения тока, протекающего по обмоткам двигателя. Управление драйвером мотора осуществляется с помощью двух цифровых логических линий: линия DIR предназначена для выбора направления вращения двигателя (высокий уровень – вращение по часовой стрелки, низкий уровень – против), на линию STEP подается ШИМ-сигнал (каждый импульс – вращение двигателя на заданную величину углового перемещения.

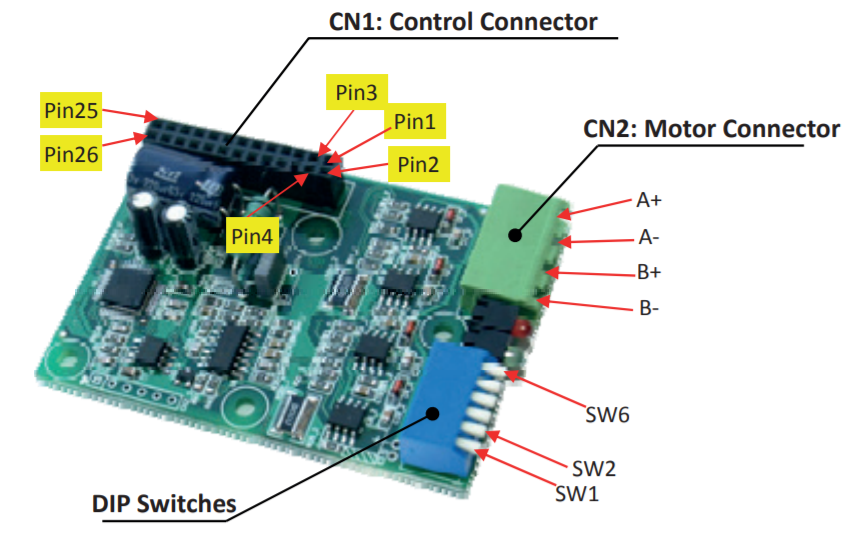


Рисунок 3 – Драйвер шагового двигателя SDM660.

* 1. **Блок для работы с ФЭУ.**

Фотоэлектронный умножитель (ФЭУ) — электровакуумный прибор, в котором поток электронов, излучаемый фотокатодом, под действием оптического излучения, усиливается в динодной системе в результате вторичной электронной эмиссии.

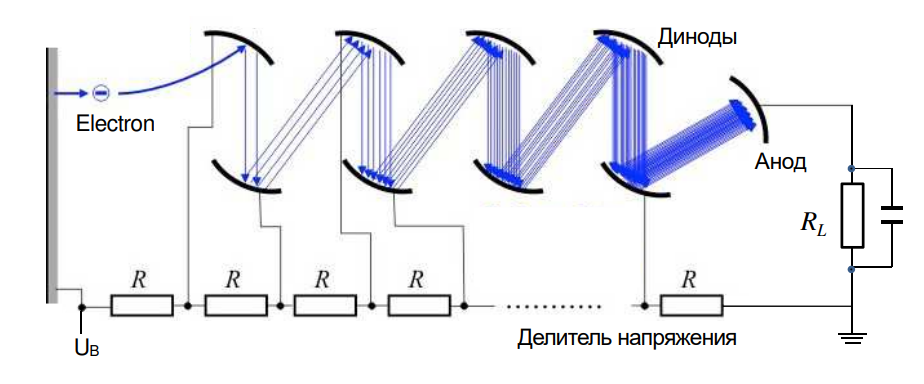


Рисунок 4 – Принцип работы ФЭУ.

На выходе ФЭУ получаем токовый сигнал, величина которого зависит от интенсивности поступающего света, а в случае подачи монохроматического излучения – от длины поступающей волны. Выходной сигнал будем усиливать и преобразовывать в напряжение с помощью операционного усилителя LMC7101, включенного в инвертирующем режиме с отрицательной обратной связью. Сигнал на выходе усилителя будет изменяться в пределах опорного напряжения ОУ (0…3,3 В). Для определения длины волны монохроматического излучения, поступающего на вход ФЭУ, полученный с выхода усилителя сигнал будем подавать на АЦП микроконтроллера STM32.

Задача по разработке блока для снятия сигнала с ФЭУ в счетном режиме не входит в данную работы. В будущем возможна разработка данного блока. Предполагаемым решением является: рассчитать усилительный каскад, подавать сигнал с каскада на триггер Шмита, считать импульсы от триггера Шмитта на STM32 в режиме подсчета входных импульсов.

* 1. **Взаимодействие с сервером.**

Серверная часть должна обеспечивать быстрое взаимодействие между пользователем и устройством. С развитием промышленного интернета вещей появилось множество промышленных протоколов передачи данных. Одним из них является MQTT (Message Queue Telemetry Transport) – легкий, компактный и открытый протокол обмена данными.

Согласно данному протоколу, обмен сообщениями осуществляется между клиентами, каждый из которых может быть как издателем, так и подписчиком. В нашем случае клиентами сети будут сервер и устройство.

Для взаимодействия устройства с сервером будем использовать Wi-Fi–модуль ESP8266–01. Данный модуль управляется 32-битным микроконтроллером ESP8266 с поддержкой Wi-Fi. Передача информации между модулем и МК STM32 будет осуществляться с помощью последовательного асинхронного интерфейса UART.

* 1. **Питание схемы.**

Питание на устройство будет подаваться +5 В. От данного напряжения схема логики драйвера мотора SDM660 для обеспечения взаимодействия с микроконтроллером STM32. Далее данное напряжение будет преобразовываться в +3,3 В с помощью линейного стабилизатора для питания: микроконтроллера STM32, Wi-Fi–модуля ESP8266–01, ОУ LMC7101 и двух реле HFO4. Схема питания имеет конденсаторы для обеспечения защиты от провалов напряжения, а также для фильтрации высокочастотных помех.

Также предусмотрен разъем для питания схемы логики драйвера мотора SDM660 +15 В и питания обмоток шагового двигателя +20…48 В.

Питание ФЭУ осуществляется подключением высоковольтного экранированного кабеля. К катоду ФЭУ прикладывается напряжение -1000 В. Для защиты МК STM32 и остальной части схемы от пробоя высокого напряжения на выходе с ОУ установлен стабилитрон VD5 и защитный диод  
VD4 на 3,3 В.

* 1. **Вывод по разделу**

Таким образом, основными элементами схемы являются:

1. STM32: реализация логики работы на стороне железа;
2. ESP8266: реализация взаимодействия по средствам Wi-Fi;
3. SDM660: драйвер для шагового двигателя, обеспечивает вращение диспергирующего элемента;
4. LMC7101: преобразования сигнала с ФЭУ для STM32;
5. LM1117-3.3: питание STM32, Wi-Fi–модуля ESP8266–01, ОУ LMC7101 и двух реле HFO4;
6. CH340G: UART-USB Driver для общения STM32, ESP6288, а так же прошивки STM32.

Серверная часть состоит из .NET севера, которых принимает данные от устройства, агрегирует их и предоставляет WEB интерфейс для пользователя.

1. **Схемотехническая реализация**

В данном разделе описаны схемотехнические решения, принятые исходя из ранее определенных параметров разрабатываемого устройства.

* 1. **STM32**

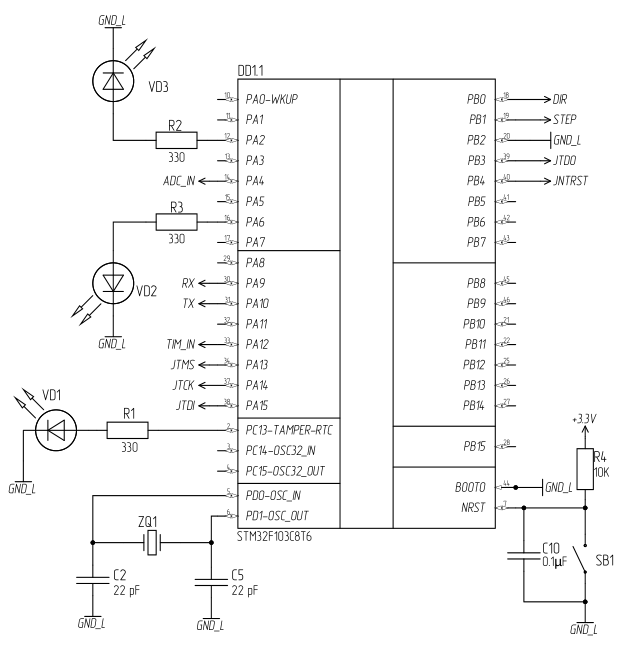


Рисунок 5 – Схема обвязки микроконтроллера STM32F103C8T6.

В качестве элемента управления ФЭУ было решено использовать 32-битный микроконтроллер STM32F103C8T6 фирмы ST Microelectronics [Рисунок 5]. Данная модель работает на тактовой частоте до 72 МГц, что обеспечивает достаточную скорость обработки входных и выходных сигналов.

В качестве источника тактирования используется кварцевый генератор ZQ1 с частотой резонанса 8 МГц. С помощью встроенных схем деления и умножения микроконтроллера данная частота преобразуется в необходимые значения тактовой частоты для различной периферии микроконтроллера. Согласно datasheet микроконтроллера, на выводы кварцевого резонатора необходимо устанавливать высокочастотные керамические конденсаторы емкостью 5…25 пФ [Рисунок 6].

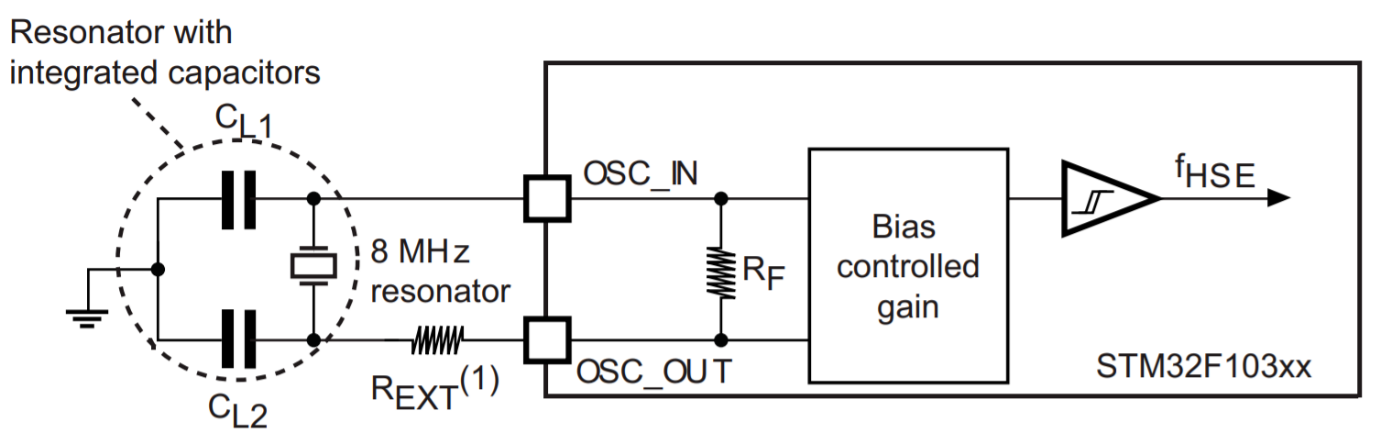


Рисунок 6 – Типичное применение кварцевого резонатора частотой 8 МГц [].

Микроконтроллер питается от источника постоянного напряжения +3,3 В [Рисунок 7]. Для фильтрации помех рядом с выводами питания устанавливаются керамические конденсаторы C1, C3, C4, C7, C9 с номинальной емкостью 0,1 мкФ.

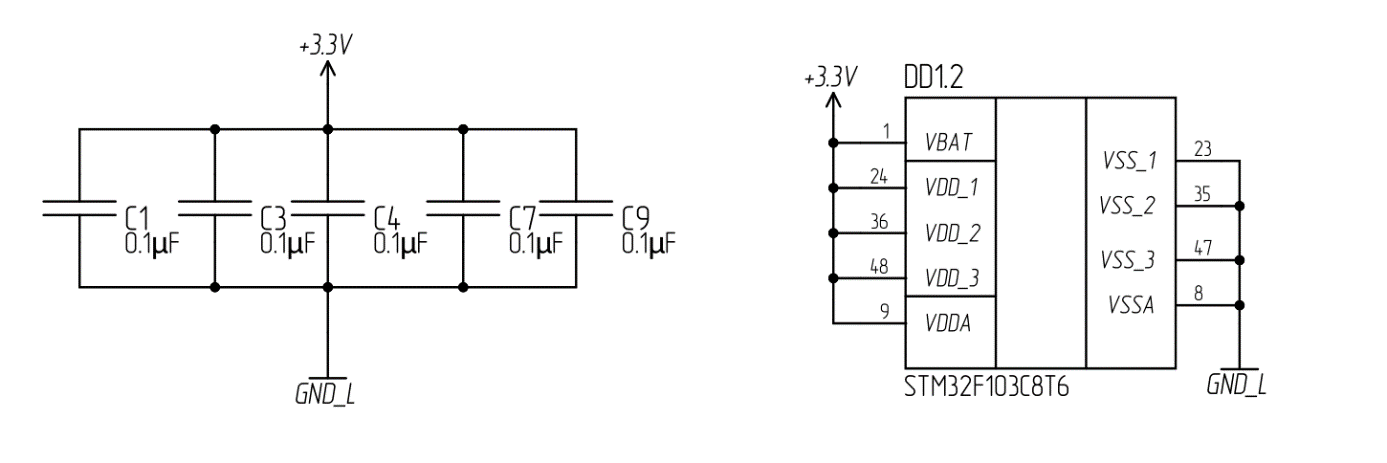


Рисунок 7 – Схема питания микроконтроллера STM32F103C8T6.

Для внешней индикации установлены три светодиода VD1, VD2 и VD3. Для ограничения тока через светодиоды установлены постоянные резисторы R1, R2 и R3 с номинальным сопротивлением 330 Ом.

Для программирования и откладки работы микроконтроллера используется интерфейс JTAG [Рисунок 8]. Все логические линии данного интерфейса подтягиваются к напряжению +3,3 В через постоянные резисторы R5, R6, R7, R8, R9 номиналом 47 кОм.

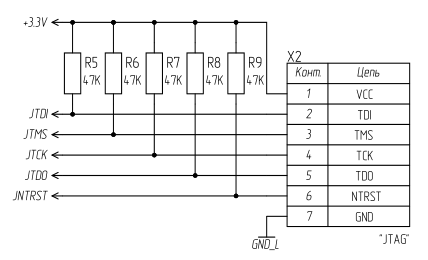


Рисунок 8 – Схема подключения интерфейса JTAG.

Выводы BOOT0 и PB2 микроконтроллера предназначены для выбора одного из трех вариантов загрузки микроконтроллера:

* Загрузка из пользовательской флеш-памяти;
* Загрузка из системной памяти;
* Загрузка из встроенного статического ОЗУ;

По умолчанию данные выводы притянуты к земле для работы с флеш-памятью микроконтроллера.

Выводы RX (PA9) и TX (PA10) являются сигнальными линиями последовательного асинхронного интерфейса передачи данных UART. Передача данных по данному интерфейсу происходит между микроконтроллером STM32 и Wi–Fi модулем ESP8266–01. Линия RX предназначена для приема данных, а TX – для передачи.

Вывод NRST предназначен для аппаратного сброса и перезагрузки микроконтроллера. В обычном режиме работы данный вывод притянут к плюсу питания через постоянный резистор номиналом 10 кОм. При необходимости сброса микроконтроллера пользователь нажимает кнопку, подтягивающую данный вывод на землю.

* 1. **ESP8266-01**

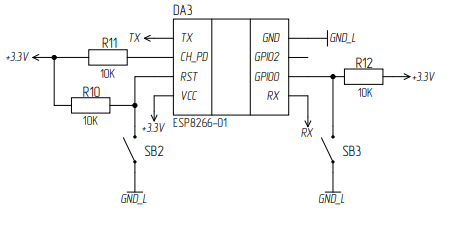


Рисунок 9 – Схема подключения модуля ESP8266–01.

Wi–Fi модуль ESP8266–01 представляет собой небольшую печатную плату с установленной обвязкой микроконтроллера ESP8266 [Рисунок 9]. Данный модуль питается от источника постоянного напряжения +3,3 В. Для фильтрации помех на входе питания установлен керамический конденсатор с номинальной емкостью 0,1 мкФ.

Вывод CH\_PD подтянут к плюсу питания через постоянный резистор R11 с номинальным сопротивлением 10 кОм. Положительный потенциал на данном входе является сигналом к работе микроконтроллера ESP8266 (Chip enable).

Вывод RST выполняет функцию, аналогичную выводу NRST микроконтроллера STM32 [2.1], и имеет схожую схему подключения.

Вывод GPIO0 устанавливает режим работы микроконтроллера. Высокий уровень – нормальная работа, низкий уровень – режим прошивки.

* 1. **SDM660**

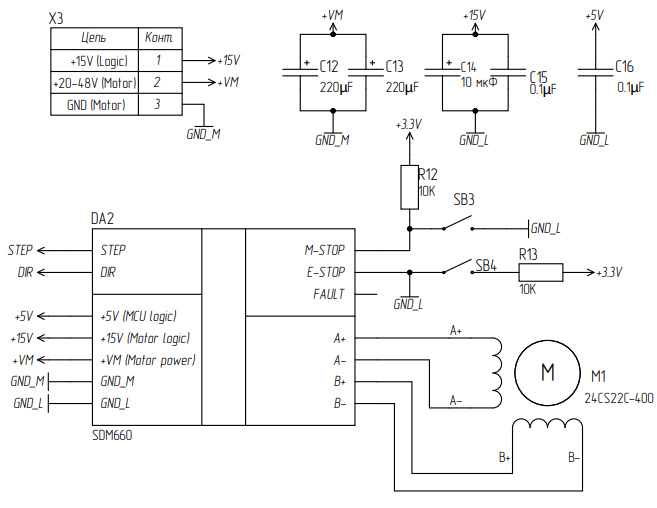


Рисунок 9 – Схема включения драйвера шагового двигателя SDM660 с питанием.

Драйвер управления шаговым двигателем 24CS22C-400 представляет собой законченное устройство в виде печатной платы [Рисунок 3]. Драйвер использует три линии питания:

* +5 В – для работы с логикой микроконтроллера;
* +15 В – для работы с внутренней логикой драйвера;
* +20…48 В – для управления обмотками шагового двигателя.

По логическим линиям питания драйвера установлены керамические конденсаторы C15 и C16 с номинальной емкостью 0,1 мкФ для фильтрации помех. По линии питания шагового двигателя установлена пара алюминиевых электролитических конденсаторов C12 и C13 с номинальной емкостью 220 мкФ для обеспечения защиты от провалов напряжения.

Выводы STEP и DIR предназначены для управления двигателем (принцип управления описан в разделе [1.1]).

Вывод M–STOP предназначен для отключения питания обмоток двигателя. Высокий уровень – на обмотки подано напряжение, двигатель имеет удерживающий момент. Низкий уровень – обмотки отключены, двигатель разблокирован. При нормальном режиме работы данный вывод подтянут к + 3,3 В через постоянный резистор с номинальным сопротивлением R12 10 кОм. С помощью кнопки с фиксацией данный вывод можно притянуть к земле для разблокировки двигателя.

Вывод E–STOP предназначен для экстренного отключения питания силовой схемы драйвера. Низкий уровень – нормальная работа силовой схемы драйвера. Высокий уровень – силовая схема драйвера отключается, питание с двигателя снимается. При нормальном режиме работы данный вывод подтянут к земле. С помощью кнопки с фиксацией данный вывод можно притянуть к +3,3 В через постоянный резистор R13 с номинальным сопротивлением 10 кОм для экстренного отключения силовой схемы.

* 1. **Блок для получения сигнала с ФЭУ.**

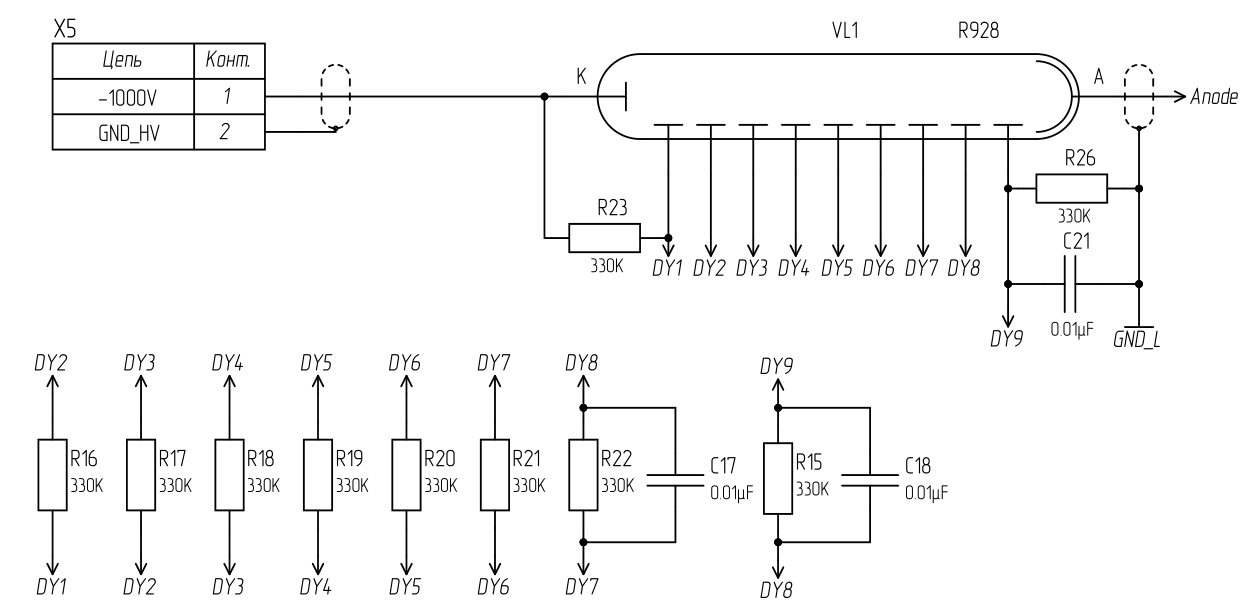


Рисунок 11 – Схема подключения ФЭУ R928.

Схема подключения ФЭУ R928 фирмы Hamamatsu разработана согласно datasheet устройства [5]. К катоду ФЭУ подводится напряжение  
 -1000 В. Между динодами устанавливаются постоянные резисторы с номинальным сопротивлением 330 кОм. На аноде ФЭУ снимаем токовый сигнал значение, которого равно Iфэу=320мА, расчитано по формуле (1).

|  |  |
| --- | --- |
|  | (1) |

Где,

g = 107 – выход электронов с ФЭУ

e = 1,6 \* 10-19 – заряд электрона

t = 5 \* 10-9 – время одного импульса

Таким образом на каждом последующем диноде будет происходить увеличение числа электронов и, согласно datasheet, из одного электрона, попавшего на первый динод, на выходе ФЭУ будет 107 электронов.

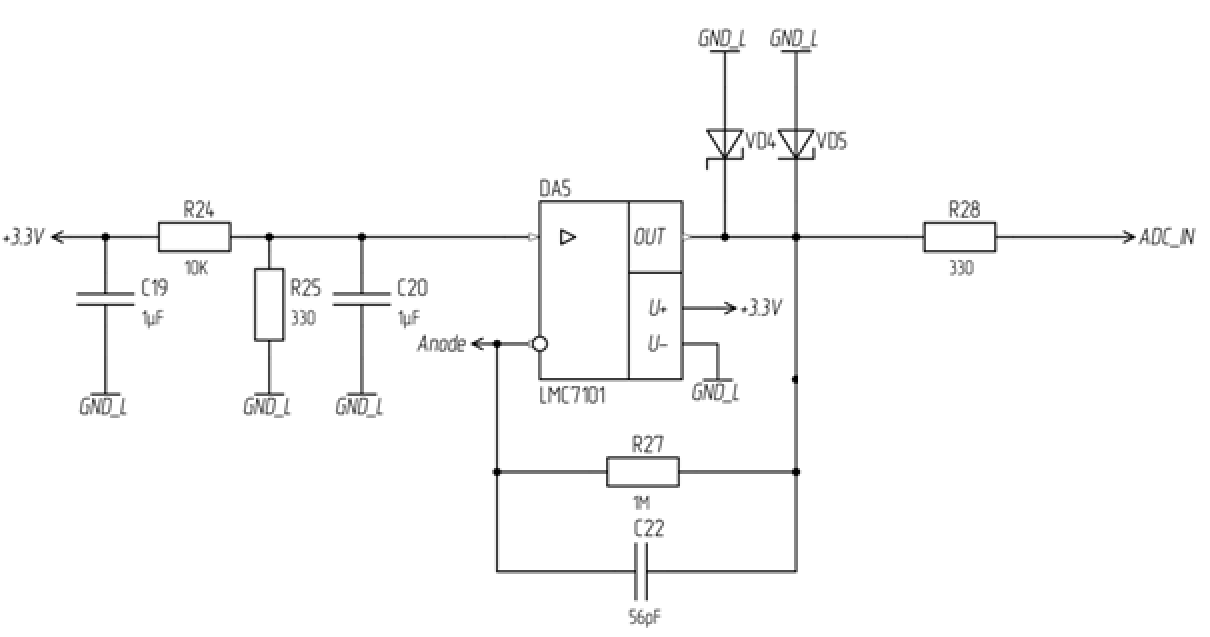


Рисунок 12 – Схема обработки выходного токового сигнала с ФЭУ.

Принцип работы данной схемы заключается в преобразовании переменных токовых сигналов с ФЭУ в напряжение с постоянным уровнем от 0 В до 3 В.

В качестве операционного усилителя используется LMC7101, его питание соответствует питанию STM32 и ESP8266, что упрощает построение питания устройства. На положительный вход подается опорное напряжение 3,3В, которое преобразуется делителем из резисторов R24 и R25 в 0,11В (2).

|  |  |
| --- | --- |
|  | (2) |

Конденсаторы C19 и C20 используются для фильтрации высокочастотных помех. Диод VD4 используется для защиты от статики и пробоя, а стабилитрон VD5 не дает напряжению подняться выше 3,3В.

Резистор R27 и C22 используются для сглаживая сигнала с ФЭУ.

Согласно datasheet на ФЭУ темновой ток IФЭУ=3нА, тогда напряжение на резисторе R27 будет 0.003В (3) данное значение попадает в разрешение канала АЦП:

|  |  |
| --- | --- |
|  | (3) |

Постоянная времени цепи RC равна 56мкс.

|  |  |
| --- | --- |
|  | (4) |

Согласно datasheet на ФЭУ rise time равно 2,2нс, таким образом множественные токовые сигналы будут сглажены в постоянный по напряжению выходной сигнал на pin STM32.

Разработка модуля для регистрации количества фотонов не входит в данную работу.

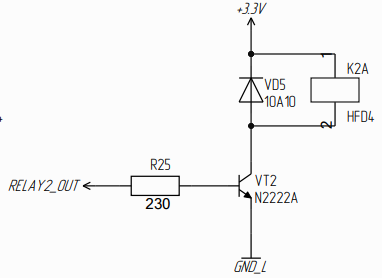


Рисунок 13 – Схема переключения реле.

Так как для обработки токового и счетного режимов с ФЭУ

используются разные периферии необходимо разработать механизм

переключения между ними. Принцип работы данного устройства

заключается в следующем: с pin STM32 подается напряжение 3.3В, согласно

datasheet ток с pin равен 10мА, транзистор N2222A VT2 переходит в стадию

насыщения, через реле HFD4 начинает протекать ток, ключ замыкается,

когда сигнал от контроллера перестает поступать, транзистор закрывается и

реле перестает пропускать ток.

Выбор элем обусловлен следующим расчетом:

RHFD4 = 64,3Ом, тогда ток, протекающий через реле при питании 3.3В равен

51мА. Максимальный ток коллектора транзистора VT2 800мА. Ток базы

равен 10мА, а напряжение на резисторе R25 равно 2.3В, тогда сопротивление

R25 будет равно 230Ом.

1. **Кодовая база**
   1. **.NET сервер**

В качестве платформы для серверного приложения был выбран .NET от компании Microsoft. Данный выбор обусловлен несколькими факторами, а именно: .NET поддерживает несколько языков программирования C#, F#, Visual Basic, имеет множество готовых инструментов для разработки WEB приложений, обновления на данную платформу выходят каждый полгода, имеется большое сообщество разработчиков, как следствие в интернете есть много информации по решению проблем, с точки зрения производительности .NET занимает лидирующие строчки во множестве рейтингов. Одни из самых важных плюсов является кроссплатформенность .NET, одно и тоже приложение можно запускать на ОС Windows, Linux и MacOS.

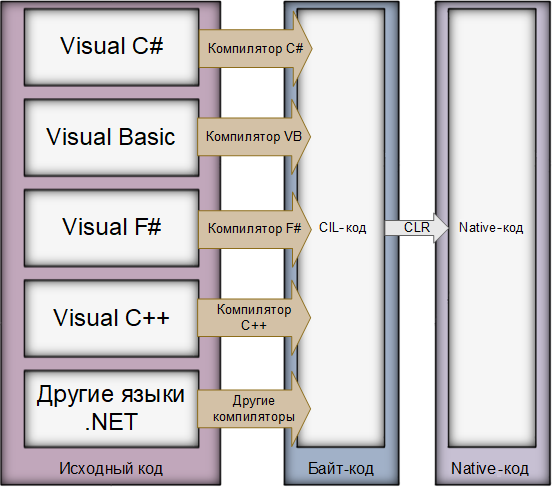
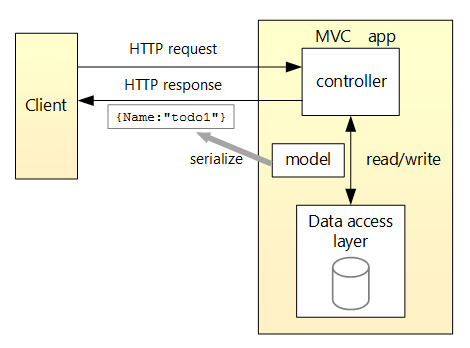


Рисунок 14 – Принцип работы .NET платформы.

Для каждого языка используется свой CLI компилятор, который преобразует программу в едины Intermediate language, затем .NET преобразует IL в байтовы коде, запускаемый на любых OS. Для компиляции кода используется SDK, содержащие драйвер dotnet, систему сбор MSBuild и набор системных библиотек. Для установки новых расширений используется единый менеджер пакетов NuGet, каждый разработчик может разместить на нем свою библиотеку для всеобщего использования.

В данном проекте использует одно .NET приложение, которое устанавливается на сервера Kestrel.

  
Рисунок 15 –Устройство .NET WEB-API.

По протоколу HTTP Client выполняет запросы на сервер, для получения данных. На сервере используется архитектура MVC (model-view-controller), где controller – отвечает за обработку HTTP запросов, model – модель данных, отправляемая пользователю, view – статический контент, как правило html, css и js для создания интерфейса взаимодействия в браузере. Для хранения данных используется база, такая как MySQL или PostgreSQL.

Для взаимодействия с ESP8266 используется протокол передачи MQTT, на сервере разворачивает дополнительный брокер сообщений, на который подписывает как ESP8266, так и WEP-API приложение. Данный выбор обусловлен: малый вес сообщений, гарантия доставки, возможность внедрения балансировки нагрузки.

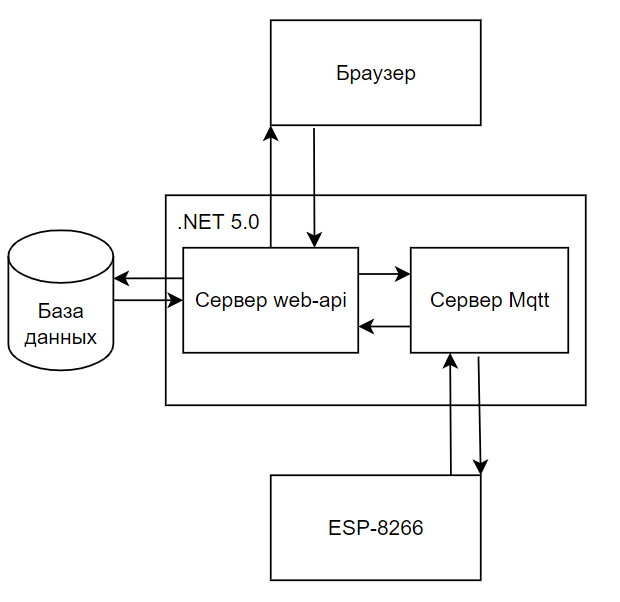


Рисунок 16 – Архитектура взаимодействия пользователя с устройством.

От сервера ожидаются следующие команды в строковом формате:

1. M\_{угол поворота в грудусах}- поворот шагового двигателя;
2. MOTOR? – узнать состояние шагового двигателя в данный момент;
3. PMT\_ADC – переключить реле в токовый режим;
4. PMT\_TIM – переключить реле в счетный режим;
5. STOP – остановить обработку данных с ФЭУ;

Сервера ожидает от устройства следующие команды в строковом формате:

1. MOTOR={угол} – угол поворота шагового двигателя в данный момент;
2. PMT- количество фотонов с ФЭУ в счетном режиме (до 106) или 12-битное значение в токовом;

C ESP8266 PMT команды отправляются каждую секунду, благодаря этому пользователь сможет в реальном времени видеть на графиках зависимость количества фотонов или люменов от длины волны.

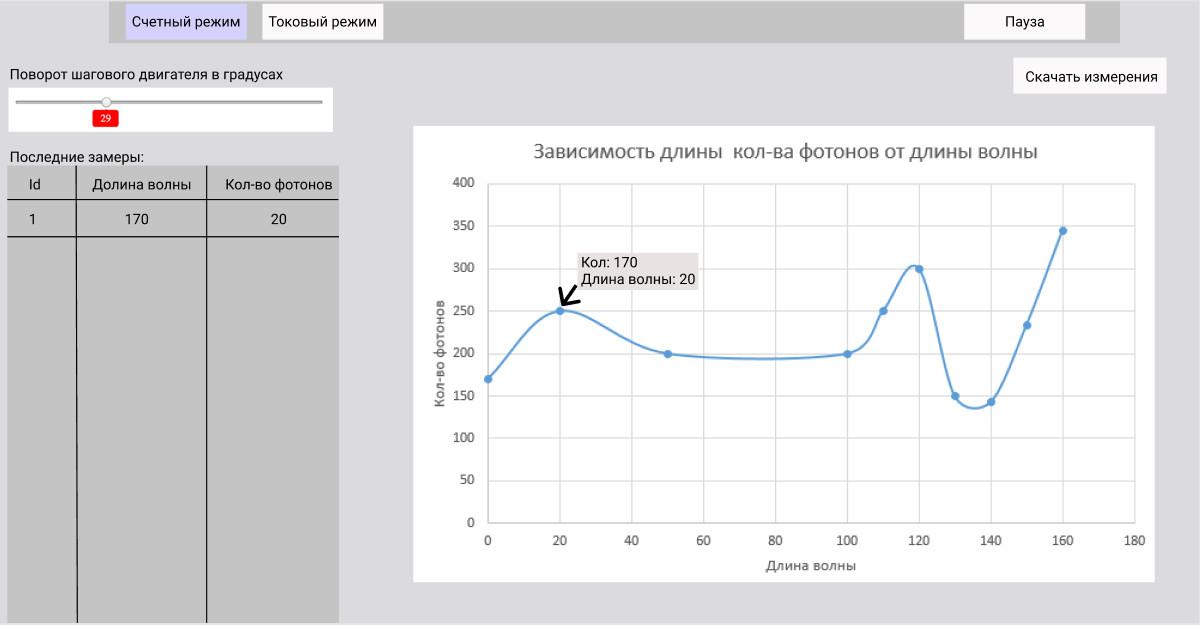


Рисунок 17 – Веб интерфейс для взаимодействия с устройством.

Данный интерфейс запускает в браузере, в нем используется html, css, js. Через Web-socket каждую секунду от сервера приходят сообщения с данными, в левой таблице подрисовываются последние измерения, они же повторяются на графике. Переключение счетного и токового режимов осуществляется в меню сверху.

**Код на STM32**

Разработка программы велась с использованием библиотеки *HAL* (англ. *Hardware Abstraction Layer*). Данная библиотека предоставляет высокоуровневые драйвера для взаимодействия с периферией микроконтроллера. Ее использование обеспечивает переносимость кода между микроконтроллерами на базе процессора *Arm Cortex-M*, а также более быстрое написание программы, поскольку вся работа с периферией микроконтроллера производится с помощью высокоуровневых функций, а не через регистры.

Для написания кода использовалось ПО *STM32Cube IDE* от компании *ST Microelectronics*. Данное приложение является универсальной платформой для разработки ПО для микроконтроллеров *STM32* на языках *C/C++*, предоставляющее возможность генерации первичного кода, компиляции, отладки и прошивки кода в память микроконтроллера.

В программе была использована многозадачная операционная система реального времени FreeRTOS. Это решение позволило разбить работу системы на несколько задач, называемых также потоками, выполняющих свою работу параллельно.

Далее описан принцип работы и алгоритм каждой из задач.

Задача *myTaskUART* предназначена для приема и парсинга данных, полученных по интерфейсу UART, а также для подачи управляющих команд остальным задачам. Алгоритм работы, следующий:

* Начальное состояние – ожидание получения символа начала пакета ‘$’;
* Получение остальной части пакета, пока не встретится символ конца ‘;’;
* Парсинг принятого сообщения – сохранение даты, времени и управляющей команды;
* Подача команды остальным задачам на исполнение принятой команды;
* Переход в начальное состояние.

Задача *myTaskMOTOR* предназначена для управления шаговым двигателем. Алгоритм работы, следующий:

* Начальное состояние – ожидание получения команды от задачи *myTaskUART* на вращение двигателя;
* Определения угла, на который необходимо повернуть двигатель, относительно текущего угла поворота;
* Определение и установка направления поворота двигателя для вращения по дуге с наименьшей длиной;
* Подача необходимого количество импульсов ШИМ сигнала на драйвер шагового двигателя для вращения;
* Переход в начальное состояние.

Задача *myTaskPMT* предназначена для сбора данных с ФЭУ. Она обеспечивает работу в двух режимах: счетном и спектрометрическом.

В спектрометрическом режиме данные поступают на вход АЦП микроконтроллера. При получении команды на включение данного режима, электромеханическое реле замыкает цепь, и сигнал поступает на вход АЦП, так же происходит размыкание реле, отвечающее за получение сигнала в счетном режиме. В течение одной секунды происходит множество замеров значений, а также фильтрация полученных данных методом «скользящего среднего». По прошествии одной секунды итоговое значение отправляется по интерфейсу UART.

При работе в счетном режиме используются аппаратные таймеры микроконтроллера. При получении команды на включение данного режима, электромеханическое реле замыкает цепь, и сигнал поступает на вход таймера, так же происходит размыкание реле отвеающего за получение . Выходной сигнал с триггера Шмитта осуществляет тактирование таймера микроконтроллера (вход *TIM2\_ETR*). Таким образом, присутствует возможность подсчета количества импульсов, поступивших на вход микроконтроллера за определенный промежуток времени. Для этого необходимо использовать как минимум 2 таймера – первый для подсчета импульсов, а второй для отсчета заданного временного промежутка.

Поскольку все таймеры микроконтроллера *STM32F103C8T6* имеют разрешение счетчика 16 бит, максимальное значение подсчитанных импульсов – 65536. Таким образом, при измерении количества импульсов за 1 секунду (частоту), максимальное их количество - 65536. Чтобы обойти это ограничение, воспользуемся еще одним встроенным таймером *TIM3*, работающим в режиме отслеживания переполнения таймера *TIM2*. Каждый раз, когда значение счетчика таймера *TIM2* будет переполняться, таймер *TIM3* будет инкрементировать счетчик. Затем, общее количество подсчитанных импульсов будет рассчитываться по формуле:

где – количество переполнений таймера *TIM2*, подсчитанных таймером *TIM3*;

– разрешение счетчика таймера;

– количество подсчитанных импульсов таймером *TIM2*.

**Код на ESP–01**

Wi–Fi модуль ESP–01 обеспечивает беспроводное соединение сервера с устройством. Программа была написана с помощью универсальной библиотеки для работы со средствами Wi-Fi. Также были использованы готовые библиотеки для установки и настройки работы модуля с сервером MQTT.

Работа программы устроена следующим образом. При включении устройства происходит инициализация: проверка параметров Wi–Fi и MQTT сервера, сохраненных в памяти микроконтроллера. Далее происходит настройка клиента, создание топиков для работы в режиме отправителя и в режиме подписчика. Затем производится настройка интерфейса UART. Далее вызывается функция подключения к серверу. Если подключение произошло безуспешно, программа пытается установить соединение каждые 5 секунд. При успешном подключении программа готова к приему и отправке данных на сервер. При получении сообщения от сервера программа отправляет его по интерфейсу UART, добавляя в начале пакета знак «$», а в конце знак «;». Аналогично, при получении сообщения по UART от микроконтроллера STM32, оно отправляется на сервер.

**Заключение**

Таким образом была разработана система из веб-сервера и устройства, подключаемого к монохроматору, для дистанционного управления и снятия результатов измерения в токовом и счетном режимах. Так же был разработан прототип веб интерфейса для пользователя.

**ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА**

* + - 1. Datasheet Photomultiplier tube R928 [Электронный ресурс]: <https://www.hamamatsu.com/resources/pdf/etd/R928_R928P_R955_R955P_TPMS1091E.pdf> (дата обращения 24.01.2022)
      2. Datasheet STM32F103x8 STM32F103xB [Электронный ресурс]: <https://www.st.com/resource/en/datasheet/stm32f103c8.pdf> (дата обращения 24.01.2022)
      3. Частотомер на микроконтроллере STM32 [Электронный ресурс]: <https://istarik.ru/blog/stm32/124.html> (дата обращения 24.01.2022)
      4. Datasheet of Leadshine SDM660 Stepper Driver Module [Электронный ресурс]: <https://purelogic.ru/docs/elektronika/SDM660d_V2.0_datasheet.pdf> (дата обращения 24.01.2022)
      5. Документация на шаговый двигатель 24CS22C-400 [Электронный ресурс]: <https://cloudray.oss-us-west-1.aliyuncs.com/%E7%94%B5%E6%9C%BA%E5%9B%BE%E7%BA%B8/24CS22C-400.PDF> (дата обращения 24.01.2022)
      6. Wi–Fi модуль ESP–01 [Электронный ресурс]: <https://3d-diy.ru/wiki/arduino-moduli/wi-fi-modul-esp-01/> (дата обращения 24.01.2022)
      7. Datasheet LMC7101, LMC7101Q-Q1 [Электронный ресурс]: <https://www.ti.com/lit/ds/symlink/lmc7101.pdf?ts=1643010106286&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLMC7101%253FHQS%253DTI-null-null-octopart-df-pf-null-wwe> (дата обращения 24.01.2022)
      8. Википедия: Фотоэлектронный умножитель [Электронный ресурс]: <https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%82%D0%BE%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D1%83%D0%BC%D0%BD%D0%BE%D0%B6%D0%B8%D1%82%D0%B5%D0%BB%D1%8C> (дата обращения 24.01.2022)

**ПРИЛОЖЕНИЕ А**

Основное тело программы *main.c*

/\* Includes \*/

#include "main.h"

#include "cmsis\_os.h"

#include "adc.h"

#include "tim.h"

#include "usart.h"

#include "gpio.h"

/\* Private function prototypes \*/

void SystemClock\_Config(void);

void MX\_FREERTOS\_Init(void);

int main(void)

{

HAL\_Init();

/\* Configure the system clock \*/

SystemClock\_Config();

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_USART1\_UART\_Init();

MX\_ADC1\_Init();

MX\_TIM1\_Init();

MX\_TIM2\_Init();

MX\_TIM3\_Init();

/\* Init scheduler \*/

osKernelInitialize(); /\* Call init function for freertos objects (in freertos.c) \*/

MX\_FREERTOS\_Init();

/\* Start scheduler \*/

osKernelStart();

/\* We should never get here as control is now taken by the scheduler \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while(1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

void SystemClock\_Config(void)

{

**Продолжение ПРИЛОЖЕНИЯ А**

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

RCC\_PeriphCLKInitTypeDef PeriphClkInit = {0};

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.HSEPredivValue = RCC\_HSE\_PREDIV\_DIV1;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL9;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK |RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

Error\_Handler();

}

PeriphClkInit.PeriphClockSelection = RCC\_PERIPHCLK\_ADC;

PeriphClkInit.AdcClockSelection = RCC\_ADCPCLK2\_DIV6;

if (HAL\_RCCEx\_PeriphCLKConfig(&PeriphClkInit) != HAL\_OK)

{

Error\_Handler();

}

}

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

while(1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

**ПРИЛОЖЕНИЕ Б**

Заголовочный файл библиотеки *CParcer.h*

/\*

\* CParser.h

\*

\* Created on: 22 окт. 2021 г.

\* Author: Дмитрий Егоров

\*/

#ifndef CPARSER\_H\_

#define CPARSER\_H\_

#pragma once

#include "main.h"

#include "usart.h"

#include "gpio.h"

#include "adc.h"

#include "string.h"

#include "stdio.h"

#include <stdlib.h>

/\*

\* Объявление типа Status, OK(ERR) будет возвращаться

\* функциями при успешном(безуспешном) выполнении

\*/

typedef enum {

OK = 0, ERR = 1

} Status;

Status receiveSymbol(); //Прием символа по UART

Status checkStartOfMessage(); //Поиск начала сообщения (символа '$')

Status parseMessage(); //Парсинг сообщения

Status controlLED(); //Управление светодиодом

Status controlFunction(); //Управление шаговым двигателем

Status transmitMessage(); //Отправить пакет по UART (для отладки)

Status receiveMessage(); //Принять сообщение по UART

uint8\_t getMessageLength(); //Получить длину принятого по UART сообщения

char LED\_State\_[20]; //Буфер для записи состояния светодиода

char PMT\_State\_[20];

char transmitBuf\_[200]; //Буфер для отправляемого сообщения (для отладки)

char receiveBuf\_[200]; //Буфер для принимаемого сообщения

char receivedSymbol\_; //Принимаемый символ по UART

**Продолжение ПРИЛОЖЕНИЯ Б**

uint8\_t messageLength\_; //Длина принятого сообщения

/\*Данные, полученные после парсинга принятого сообщения\*/

char date\_[15]; //Дата

char time\_[15]; //Время

char payload\_[15]; //Данные

uint8\_t oldMotorRotationAngle\_; //Прежнее значение угла поворота

uint8\_t newMotorRotationAngle\_; //Новое значение угла поворота

uint8\_t MOTOR\_StartFlag\_; //Флаг старта работы с двигателем

uint8\_t ADC\_StartFlag\_; //Флаг старта работы с АЦП

uint8\_t TIM\_StartFlag\_; //Флаг старта работы с таймерами

int freq\_; //Полученная частота

#endif /\* CPARSER\_H\_ \*/

**ПРИЛОЖЕНИЕ В**

Исходный файл библиотеки *CParcer.c*

/\*

\* CParser.c

\*

\* Created on: 22 окт. 2021 г.

\* Author: Дмитрий Егоров

\*/

#include "CParser.h"

Status receiveSymbol() {

if (HAL\_UART\_Receive(&huart1, (uint8\_t\*) &receivedSymbol\_, 1,

HAL\_MAX\_DELAY) == HAL\_OK)

return OK;

else

return ERR;

}

/\*

\* Если принятый символ '$' (начало пакета), то возвращаем OK, иначе ERR

\*/

Status checkStartOfMessage() {

if (receivedSymbol\_ == '$') {

return OK;

} else

return ERR;

}

/\*

\* Читаем символы из UART, пока не встретим ';' (конец пакета),

\* в конце ставим символ конца строки и возвращаем OK

\* Если произошла ошибка при чтении, возвращаем ERR

\*/

Status receiveMessage() {

uint8\_t i = 0;

do {

if (receiveSymbol() == OK) {

receiveBuf\_[i] = receivedSymbol\_;

i++;

} else

return ERR;

} while (receivedSymbol\_ != ';');

receiveBuf\_[i] = '\0';

messageLength\_ = i;

return OK;

}

/\*

\* Из полученного сообщения выхватываем дату, время

**Продолжение ПРИЛОЖЕНИЯ В**

\* и управляющую команду, раскидываем данные по

\* соответствующим полям

\*/

Status parseMessage() {

uint8\_t j = 0;

for (uint8\_t k = 0; k < 10; k++) {

date\_[j] = receiveBuf\_[k];

j++;

}

date\_[10] = '\0';

j = 0;

for (uint8\_t k = 11; k < 19; k++) {

time\_[j] = receiveBuf\_[k];

j++;

}

time\_[8] = '\0';

j = 33;

uint8\_t i = 0;

while (receiveBuf\_[j] != ';') {

payload\_[i] = receiveBuf\_[j];

i++;

j++;

}

payload\_[i] = '\0';

return OK;

}

/\*

\* В зависимости от полученной команды меняем состояние светодиода

\* и записываем его состояние для дальнейшей отправки по UART

\*/

Status controlLED() {

if (!strcmp(payload\_, "LED1:1\0")) //Включение 1-го светодиода

{

HAL\_GPIO\_WritePin(LED1\_GPIO\_Port, LED1\_Pin, GPIO\_PIN\_SET);

sprintf(LED\_State\_, "LED1 State: ON%c", '\0');

} else if (!strcmp(payload\_, "LED1:0\0")) //Выключение 1-го светодиода

{

HAL\_GPIO\_WritePin(LED1\_GPIO\_Port, LED1\_Pin, GPIO\_PIN\_RESET);

sprintf(LED\_State\_, "LED1 State: OFF%c", '\0');

} else if (!strcmp(payload\_, "LED2:1\0")) //Включение 2-го светодиода

{

HAL\_GPIO\_WritePin(LED2\_GPIO\_Port, LED2\_Pin, GPIO\_PIN\_SET);

sprintf(LED\_State\_, "LED2 State: ON%c", '\0');

**Продолжение ПРИЛОЖЕНИЯ В**

} else if (!strcmp(payload\_, "LED2:0\0")) //Выключение 2-го светодиода

{

HAL\_GPIO\_WritePin(LED2\_GPIO\_Port, LED2\_Pin, GPIO\_PIN\_RESET);

sprintf(LED\_State\_, "LED2 State: OFF%c", '\0');

} else if (!strcmp(payload\_, "LED?\0")) //Запрос состояния светодиодов

{

uint8\_t led1state = (uint8\_t) HAL\_GPIO\_ReadPin(LED1\_GPIO\_Port,

LED1\_Pin);

uint8\_t led2state = (uint8\_t) HAL\_GPIO\_ReadPin(LED2\_GPIO\_Port,

LED2\_Pin);

sprintf(LED\_State\_, "LED1:%d, LED2:%d%c", led1state, led2state, '\0');

if (HAL\_UART\_Transmit(&huart1, (uint8\_t\*) &LED\_State\_,

strlen(LED\_State\_), 1000) == HAL\_OK) {

} else

return ERR;

}

return OK;

}

Status controlFunction() {

if (!strcmp(payload\_, "MOTOR?\0")) {

if (HAL\_UART\_Transmit(&huart1, (uint8\_t\*) &newMotorRotationAngle\_,

strlen(LED\_State\_), 1000) == HAL\_OK) {

} else

return ERR;

} else if (payload\_[0] == 'M' && payload\_[1] == '\_') {

if (strlen(payload\_) == 3) {

newMotorRotationAngle\_ = (payload\_[2] - '0');

} else if (strlen(payload\_) == 4) {

newMotorRotationAngle\_ = 10 \* (payload\_[2] - '0')

+ (payload\_[3] - '0');

} else if (strlen(payload\_) == 5) {

newMotorRotationAngle\_ = 100 \* (payload\_[2] - '0')

+ 10 \* (payload\_[3] - '0') + (payload\_[4] - '0');

}

if (newMotorRotationAngle\_ == oldMotorRotationAngle\_) {

if (HAL\_UART\_Transmit(&huart1,

(uint8\_t\*) "You've transmitted the same rotation angle",

strlen("You've transmitted the same rotation angle"), 1000)

**Продолжение ПРИЛОЖЕНИЯ В**

== HAL\_OK) {

} else

return ERR;

return ERR;

} else if (newMotorRotationAngle\_ < 0 || newMotorRotationAngle\_ > 360) {

if (HAL\_UART\_Transmit(&huart1,

(uint8\_t\*) "You've transmitted the wrong rotation angle",

strlen("You've transmitted the wrong rotation angle"), 1000)

== HAL\_OK) {

} else

return ERR;

return ERR;

} else MOTOR\_StartFlag\_ = 1;

} else if (!strcmp(payload\_, "PMT\_ADC\0")) {

ADC\_StartFlag\_ = 1;

TIM\_StartFlag\_ = 0;

} else if (!strcmp(payload\_, "PMT\_TIM\0")) {

TIM\_StartFlag\_ = 1;

ADC\_StartFlag\_ = 0;

} else if (!strcmp(payload\_, "STOP\0")) {

ADC\_StartFlag\_ = 0;

TIM\_StartFlag\_ = 0;

}

return OK;

}

/\*

\* Формируем и отправляем сообщение для отправки по UART (для отладки)

\* Если отправилось, возвращаем OK, иначе ERR

\*/

Status transmitMessage() {

sprintf(transmitBuf\_,

"\r\nReceived data:\r\nDate: %s\r\nTime: %s\r\nPayload: %s\r\n%s",

date\_, time\_, payload\_, LED\_State\_);

if (HAL\_UART\_Transmit(&huart1, (uint8\_t\*) &transmitBuf\_, getMessageLength(),

1000) == HAL\_OK) {

return OK;

} else

return ERR;

}

/\*

\* Возвращаем длину полученного сообщения

\*/

uint8\_t getMessageLength() {

return strlen(transmitBuf\_);

}

**ПРИЛОЖЕНИЕ Г**

Исходный файл работы с ОС *freertos.c*

/\* Includes \*/

#include "FreeRTOS.h"

#include "task.h"

#include "main.h"

#include "cmsis\_os.h"

/\* Private includes \*/

#include "CParser.h"

#include "queue.h"

#include "tim.h"

osThreadId\_t defaultTaskHandle;

const osThreadAttr\_t defaultTask\_attributes = { .name = "defaultTask",

.stack\_size = 128 \* 4, .priority = (osPriority\_t) osPriorityNormal, };

/\* Definitions for myTaskUART \*/

osThreadId\_t myTaskUARTHandle;

const osThreadAttr\_t myTaskUART\_attributes = { .name = "myTaskUART",

.stack\_size = 128 \* 4, .priority = (osPriority\_t) osPriorityNormal, };

/\* Definitions for myTaskMotor \*/

osThreadId\_t myTaskMotorHandle;

const osThreadAttr\_t myTaskMotor\_attributes = { .name = "myTaskMotor",

.stack\_size = 128 \* 4, .priority = (osPriority\_t) osPriorityNormal, };

/\* Definitions for myTaskPMT \*/

osThreadId\_t myTaskPMTHandle;

const osThreadAttr\_t myTaskPMT\_attributes = { .name = "myTaskPMT", .stack\_size =

128 \* 4, .priority = (osPriority\_t) osPriorityNormal, };

/\* Private function prototypes -----------------------------------------------\*/

/\*

\* Эта функция нужна для отладки кода. При вызове

\* отправляет по UART сообщение об ошибке с путем к файлу

\* и номером строки, где данная функция была вызвана

\*/

void Error\_Message(uint8\_t \*file, uint32\_t line) {

char buf[200] = { 0 };

sprintf(buf, "\r Exception: Wrong parameters value: file %s on line %d\r\n",

file, (int) line);

while (HAL\_UART\_Transmit(&huart1, (uint8\_t\*) buf, strlen(buf),

10 \* strlen(buf)) != HAL\_OK)

**Продолжение ПРИЛОЖЕНИЯ Г**

;

int tick = HAL\_GetTick();

while ((HAL\_GetTick() - tick) < 5000) {

HAL\_GPIO\_TogglePin(LED\_GPIO\_Port, LED\_Pin);

HAL\_Delay(100);

}

HAL\_NVIC\_SystemReset();

}

void StartDefaultTask(void \*argument);

void StartTaskUART(void \*argument);

void StartTaskMOTOR(void \*argument);

void StartTaskPMT(void \*argument);

void MX\_FREERTOS\_Init(void); /\* (MISRA C 2004 rule 8.1) \*/

/\* Hook prototypes \*/

void vApplicationStackOverflowHook(xTaskHandle xTask, signed char \*pcTaskName);

/\* USER CODE BEGIN 4 \*/

void vApplicationStackOverflowHook(xTaskHandle xTask, signed char \*pcTaskName) {

/\* Run time stack overflow checking is performed if

configCHECK\_FOR\_STACK\_OVERFLOW is defined to 1 or 2. This hook function is

called if a stack overflow is detected. \*/

}

void MX\_FREERTOS\_Init(void) {

/\* Create the thread(s) \*/

/\* creation of defaultTask \*/

defaultTaskHandle = osThreadNew(StartDefaultTask, NULL,

&defaultTask\_attributes);

/\* creation of myTaskUART \*/

myTaskUARTHandle = osThreadNew(StartTaskUART, NULL, &myTaskUART\_attributes);

/\* creation of myTaskMotor \*/

myTaskMotorHandle = osThreadNew(StartTaskMOTOR, NULL,

&myTaskMotor\_attributes);

/\* creation of myTaskPMT \*/

myTaskPMTHandle = osThreadNew(StartTaskPMT, NULL, &myTaskPMT\_attributes);

}

void StartDefaultTask(void \*argument) {

/\* USER CODE BEGIN StartDefaultTask \*/

**Продолжение ПРИЛОЖЕНИЯ Г**

/\* Infinite loop \*/

for (;;) {

osDelay(1);

}

/\* USER CODE END StartDefaultTask \*/

}

void StartTaskUART(void \*argument) {

/\* USER CODE BEGIN StartTaskUART \*/

/\* Infinite loop \*/

for (;;) {

if (receiveSymbol() == OK) {

if (checkStartOfMessage() == OK) {

if (receiveMessage() == OK) {

if (parseMessage() == OK) {

if (controlFunction() == OK) {

}

} else

Error\_Message((uint8\_t\*) \_\_FILE\_\_, \_\_LINE\_\_);

} else

Error\_Message((uint8\_t\*) \_\_FILE\_\_, \_\_LINE\_\_);

} else

Error\_Message((uint8\_t\*) \_\_FILE\_\_, \_\_LINE\_\_);

} else

Error\_Message((uint8\_t\*) \_\_FILE\_\_, \_\_LINE\_\_);

osDelay(1);

}

/\* USER CODE END StartTaskUART \*/

}

void StartTaskMOTOR(void \*argument) {

/\* USER CODE BEGIN StartTaskMOTOR \*/

MOTOR\_StartFlag\_ = 0;

int16\_t diff = 0;

/\* Infinite loop \*/

for (;;) {

/\*

\* В зависимости от значения принятого угла относительно

\* текущего выставляем пин DIRECTION, подаем питание на пин ENABLE

\* и подаем импульсы на пин STEP для вращения

\*/

if (MOTOR\_StartFlag\_) {

diff = newMotorRotationAngle\_ - oldMotorRotationAngle\_;

if (diff != 0 && abs(diff) != 200) {

**Продолжение ПРИЛОЖЕНИЯ Г**

if (diff < -100) {

diff += 200;

HAL\_GPIO\_WritePin(MOTOR\_Port, DIR\_Pin, GPIO\_PIN\_RESET);

osDelayUntil(2);

} else if (diff > 100) {

diff = 200 - diff;

HAL\_GPIO\_WritePin(MOTOR\_Port, DIR\_Pin, GPIO\_PIN\_SET);

osDelayUntil(2);

} else if (diff > 0 && diff <= 100) {

HAL\_GPIO\_WritePin(MOTOR\_Port, DIR\_Pin, GPIO\_PIN\_SET);

osDelayUntil(2);

} else if (diff < 0 && diff >= -100) {

HAL\_GPIO\_WritePin(MOTOR\_Port, DIR\_Pin, GPIO\_PIN\_RESET);

osDelayUntil(2);

}

for (uint8\_t i = 0; i < diff; ++i) {

HAL\_GPIO\_WritePin(MOTOR\_Port, STEP\_Pin, GPIO\_PIN\_SET);

osDelayUntil(2);

HAL\_GPIO\_WritePin(MOTOR\_Port, STEP\_Pin, GPIO\_PIN\_RESET);

osDelayUntil(2);

}

HAL\_GPIO\_WritePin(MOTOR\_Port, DIR\_Pin, GPIO\_PIN\_RESET);

osDelayUntil(2);

/\*

\* Сохраняем значение нового угла поворота

\*/

oldMotorRotationAngle\_ = newMotorRotationAngle\_;

MOTOR\_StartFlag\_ = 0;

}

}

osDelay(1);

}

/\* USER CODE END StartTaskMOTOR \*/

}

void StartTaskPMT(void \*argument) {

/\* USER CODE BEGIN StartTaskPMT \*/

uint16\_t ADC\_value = 0;

uint8\_t averageFactor = 0;

uint16\_t ADC\_oldValue = 0;

HAL\_ADCEx\_Calibration\_Start(&hadc1);

/\* Infinite loop \*/

**Продолжение ПРИЛОЖЕНИЯ Г**

for (;;) {

if (ADC\_StartFlag\_) {

HAL\_GPIO\_WritePin(Relay\_OUT\_GPIO\_Port, Relay\_OUT\_Pin, GPIO\_PIN\_SET);

uint32\_t currentTime = HAL\_GetTick();

do {

HAL\_ADC\_Start(&hadc1);

ADC\_value = (uint16\_t) HAL\_ADC\_GetValue(&hadc1);

HAL\_ADC\_Stop(&hadc1);

if (averageFactor > 0) {

ADC\_oldValue = ADC\_value;

ADC\_value = (uint16\_t) ((ADC\_oldValue \* (averageFactor - 1)

+ ADC\_value) / averageFactor);

}

} while (HAL\_GetTick() - currentTime < 1000);

sprintf(PMT\_State\_, "PMT\_ADC=%d%c", ADC\_value, '\0');

if (HAL\_UART\_Transmit(&huart1, (uint8\_t\*) &PMT\_State\_,

strlen(PMT\_State\_), 1000) == HAL\_OK) {

}

} else if (TIM\_StartFlag\_) {

HAL\_TIM\_Base\_Start(&htim2);

HAL\_TIM\_Base\_Start(&htim3);

HAL\_TIM\_Base\_Start\_IT(&htim1);

sprintf(PMT\_State\_, "PMT\_TIM=%d%c", freq\_, '\0');

if (HAL\_UART\_Transmit(&huart1, (uint8\_t\*) &PMT\_State\_,

strlen(PMT\_State\_), 1000) == HAL\_OK) {

}

}

osDelay(1);

}

/\* USER CODE END StartTaskPMT \*/

}

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

//Функция-обработчик прерываний таймеров

{

if (htim == &htim1) {

uint16\_t TIM2\_count = \_\_HAL\_TIM\_GET\_COUNTER(&htim2);

//Значение счетчика таймера TIM2

//(Количество последних подсчитанных

//импульсов к моменту прерывания)

uint16\_t TIM2\_overflows = \_\_HAL\_TIM\_GET\_COUNTER(&htim3);

//Значение счетчика таймера TIM3

freq\_ = TIM2\_count + (TIM2\_overflows \* 65535) + TIM2\_overflows;

**Продолжение ПРИЛОЖЕНИЯ Г**

//Вычисление частоты

//Установка флага окончания измерения частоты

\_\_HAL\_TIM\_SET\_COUNTER(&htim2, 0x0000);

//Обнуление счетчиков таймеров 2 и 3

\_\_HAL\_TIM\_SET\_COUNTER(&htim3, 0x0000);

}

}

**ПРИЛОЖЕНИЕ Д**

Исходный файл прошивки модуля ESP–01

#include <ESP8266WiFi.h>

#include "Adafruit\_MQTT.h"

#include "Adafruit\_MQTT\_Client.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Параметры WiFi \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define WLAN\_SSID "MTSRouter-008073" //SSID точки доступа WiFi

#define WLAN\_PASS "××××××××" //Пароль к точке доступа

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Параметры MQTT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SERVER "192.168.1.2" //Адрес сервера

#define SERVERPORT 1883 //Порт

#define SERVERID "test" //ID сервера

#define USERNAME "bud" //Имя пользователя

#define PASSWORD "×××××××××" //Пароль

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Настройка клиента \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Создаем объект класса WiFiClient для подключения к WiFi

WiFiClient client;

// Настройка клиента MQTT

Adafruit\_MQTT\_Client mqtt(&client, SERVER, SERVERPORT, SERVERID, USERNAME, PASSWORD);

// Настройка отправителя

Adafruit\_MQTT\_Publish publisher = Adafruit\_MQTT\_Publish(&mqtt, "fromesp");

// Настройка подписчика

Adafruit\_MQTT\_Subscribe subscriber = Adafruit\_MQTT\_Subscribe(&mqtt, "toesp");

// Прототип функции подключения к серверу

void MQTT\_connect();

// Настройка UART (Serial), WiFi и подключение к нему, запуск подписчика

void setup() {

Serial.begin(115200);

delay(10);

Serial.println(F("Adafruit MQTT demo"));

**Продолжение ПРИЛОЖЕНИЯ Д**

Serial.println(); Serial.println();

Serial.print("Connecting to ");

Serial.println(WLAN\_SSID);

WiFi.begin(WLAN\_SSID, WLAN\_PASS);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println();

Serial.println("WiFi connected");

Serial.println("IP address: "); Serial.println(WiFi.localIP());

mqtt.subscribe(&subscriber);

}

void loop() {

MQTT\_connect();

Adafruit\_MQTT\_Subscribe \*subscription;

char buf[16]; // Буфер для сообщения состояния светодиодов

while ((subscription = mqtt.readSubscription(10000))) { // Ждем принятия сообщения с сервера

if (subscription == &subscriber) {

// Отправляем принятое сообщение по UART, добавляя в начале символ начала сообщения '$',

// а в конце ';'

Serial.print('$');

Serial.print((char \*)subscriber.lastread);

Serial.println(';');

// Читаем из UART 15 символов, записывая в буфер

Serial.readBytes(buf, 15);

// Отправляем полученный буфер на сервер

// и очищаем его, чтобы повторно не зайти в условие, не получив сообщения

if (strstr(buf, "PMT"))

{

publisher.publish((uint8\_t\*)&buf, 15);

memset(buf, '0', 15);

}

}

}

}

// Функция подключения к MQTT серверу

void MQTT\_connect() {

**Продолжение ПРИЛОЖЕНИЯ Д**

int8\_t ret;

// Если подключено, то выходим из функции

if (mqtt.connected()) {

return;

}

// Если не подключено, то пытаемся подключиться каждые 5 секунд

Serial.print("Connecting to MQTT... ");

while ((ret = mqtt.connect()) != 0) { // connect() возвращает 0, если подключено

Serial.println(mqtt.connectErrorString(ret));

Serial.println("Retrying MQTT connection in 5 seconds...");

mqtt.disconnect();

delay(5000); // wait 5 seconds

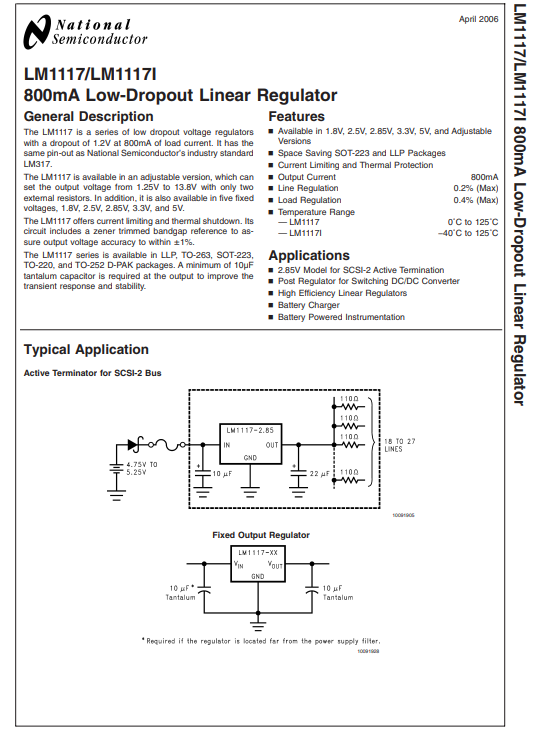
}

Serial.println("MQTT Connected!");

}

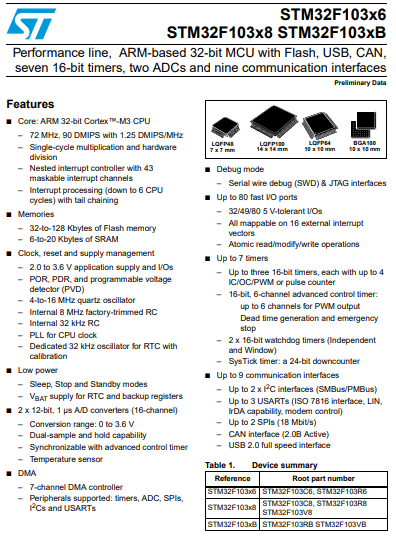
# ПРИЛОЖЕНИЕ Е

Основные характеристики микросхемы LM1117[1]

****

# ПРИЛОЖЕНИЕ Ж

Основные характеристики STM32F103[2]



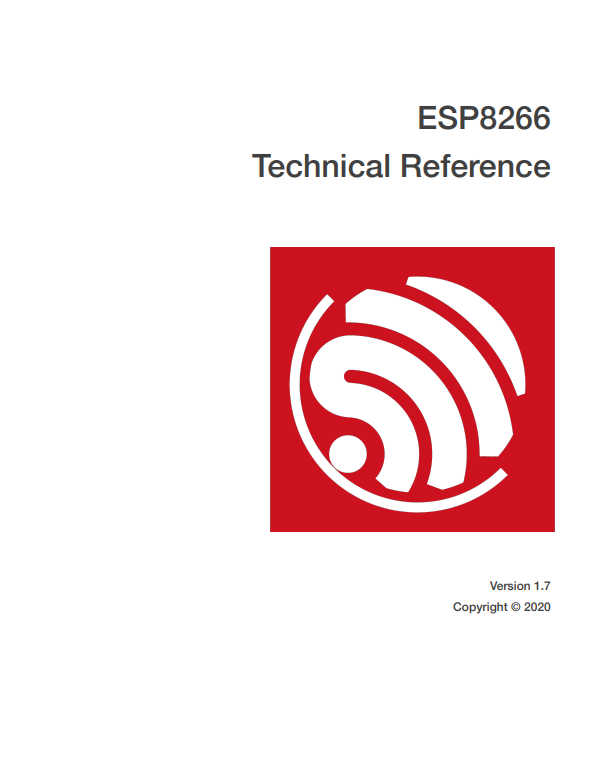
# ПРИЛОЖЕНИЕ З

Основные характеристики SDM660[3]



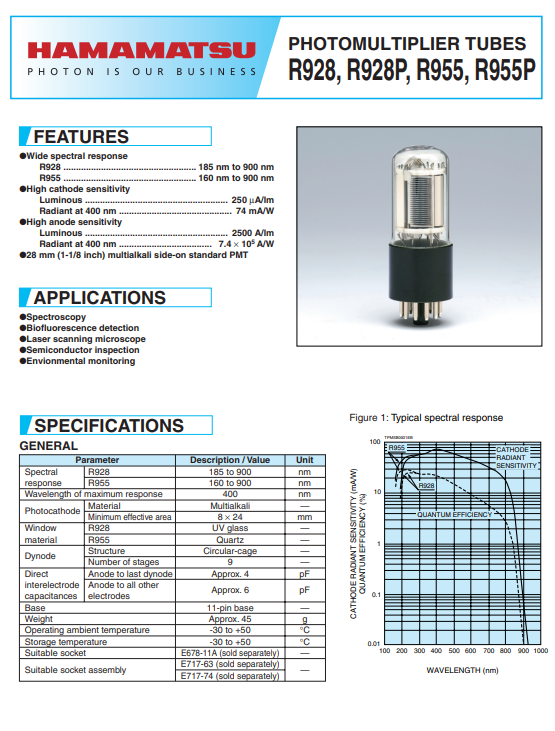
# ПРИЛОЖЕНИЕ И

Основные характеристики ESP8266[4]



# ПРИЛОЖЕНИЕ Й

# Основные характеристики R928[5]



# ПРИЛОЖЕНИЕ К

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Перв. примен.* |  | *Поз.*  *обозначение* | | | *Наименование* | | | | *Кол-во* | | *Примечание* | | | | | |
|  | | | ***Конденсаторы*** | | | |  | |  | | | | | |
| *C1* | | | *GRM188R71H104K (0.1 мкФ 50 В SMD 0603)* | | | | *1* | | *Murata* | | | | | |
| *C2* | | | *GRM1885C1H220GA01D (22 нФ 50 В SMD 0603)* | | | | *1* | | *Murata* | | | | | |
| *С3-C4* | | | *GRM188R71H104K (0.1 мкФ 50 В SMD 0603)* | | | | *2* | | *Murata* | | | | | |
| *С5* | | | *GRM1885C1H220GA01D (22 нФ 50 В SMD 0603)* | | | | *1* | | *Murata* | | | | | |
| *C6* | | | *JCK1H100M063054 (10 мкФ 50 В SMD 6.3x5.4)* | | | | *1* | | *JB* | | | | | |
| *C7-C9* | | | *GRM188R71H104K (0.1 мкФ 50 В SMD 0603)* | | | | *2* | | *Murata* | | | | | |
| *C10* | | | *JCK1H100M063054 (10 мкФ 50 В SMD 6.3x5.4)* | | | | *1* | | *JB* | | | | | |
| *C11* | | | *GRM188R71H104K (0.1 мкФ 50 В SMD 0603)* | | | | *1* | | *Murata* | | | | | |
| *C12- C13* | | | *JCK1H221M100105 (220 мкФ 50 В SMD 10x10.5)* | | | | *2* | | *JB* | | | | | |
| *C14* | | | *JCK1H100M063054 (10 мкФ 50 В SMD 6.3x5.4)* | | | | *1* | | *JB* | | | | | |
| *C15- C16* | | | *GRM188R71H104K (0.1 мкФ 50 В SMD 0603)* | | | | *2* | | *Murata* | | | | | |
| *C17-C19* | | | *GRM188R71H103K (0.01 мкФ 50 В SMD 0603)* | | | | *1* | | *Murata* | | | | | |
| *C20-C21* | | | *GRM188R71H104K (0.1 мкФ 50 В SMD 0603)* | | | | *2* | | *Murata* | | | | | |
| *M1* | | | *Шаговый двигатель 24CS22C-400* | | | | *1* | | *Cloudray* | | | | | |
| *Справ. №* |  |  | | |  | | | |  | |  | | | | | |
|  | | | ***Резисторы*** | | | |  | |  | | | | | |
| *R1-R3* | | | *Р1-8В-0603-330 Ом-±5%-50 В* | | | | *3* | | *НПО «Эркон»* | | | | | |
| *R4* | | | *Р1-8В-0603-10 кОм-±5%-50 В* | | | | *1* | | *НПО «Эркон»* | | | | | |
| *R5-R9* | | | *Р1-8В-0603-47 кОм-±5%-50 В* | | | | *5* | | *НПО «Эркон»* | | | | | |
| *R10-R14* | | | *Р1-8В-0603-10 кОм-±5%-50 В* | | | | *5* | | *НПО «Эркон»* | | | | | |
| *R15-R18* | | | *Р1-8В-0603-330 кОм-±5%-50 В* | | | | *4* | | *НПО «Эркон»* | | | | | |
| *R19* | | | *Р1-8В-0603-3650 Ом-±5%-50 В* | | | | *1* | | *НПО «Эркон»* | | | | | |
| *R20-R24* | | | *Р1-8В-0603-330 кОм-±5%-50 В* | | | | *5* | | *НПО «Эркон»* | | | | | |
|  |  |  | |  |  | *УРФУ.000000.001 ПЭ3* | | | | | | | | |
|  |  |  | |  |  |
| *Инв. № подл.* |  | *Изм.* | *Лист* | *№ докум.* | | *Подп.* | *Дата* |
| *Разраб.* | | *Егоров Д. С,* | |  |  |  | | *Лит.* | | | | *Лист* | | *Листов* |
| *Пров.* | | *Ищенко А. В.* | |  |  |  | |  |  | | *1* | *3* |
| *Нач.сект.* | |  | |  |  | *Перечень элементов* | |  | | | | | | |
| *Н.контр.* | |  | |  |  |  | | | | | | |
| *Утв.* | |  | |  |  |  | |  | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | *Поз. обозна-чение* | | | *Наименование* | | | | *Кол.* | *Примечание* | |
| *R25* | | | *Р1-8В-0603-3650 Ом-±5%-50 В* | | | | *1* | *НПО «Эркон»* | |
| *R26* | | | *Р1-8В-0603-330 кОм-±5%-50 В* | | | | *1* | *НПО «Эркон»* | |
| *R27* | | | *Р1-8В-0603-10 кОм-±5%-50 В* | | | | *1* | *НПО «Эркон»* | |
| *R28* | | | *Р1-8В-0603-330 Ом-±5%-50 В* | | | | *1* | *НПО «Эркон»* | |
| *R29* | | | *Р1-8В-0603-1 МОм-±5%-50 В* | | | | *1* | *НПО «Эркон»* | |
| *R30* | | | *Р1-8В-0603-330 Ом-±5%-50 В* | | | | *1* | *НПО «Эркон»* | |
|  | | |  | | | |  |  | |
|  | | | ***Диоды*** | | | |  |  | |
| *VD1-VD3* | | | *BL-LS0805DNB* | | | | *3* | *Betlux* | |
| *VD4-VD5* | | | *10A10* | | | | *2* | *Китай* | |
|  | | |  | | | |  |  | |
| *VL1* | | | *Прибор электровакуумный R928* | | | | *1* | *Hamamatsu* | |
| *VT1-VT2* | | | *Транзистор N2222A* | | | | *2* | *ON Semi* | |
|  | | |  | | | |  |  | |
|  | | | ***Соединения контактные*** | | | |  |  | |
| *X1* | | | *DS1095-05* | | | | *1* | *Connfly* | |
| *X2* | | | *IDC-10MR* | | | | *1* | *Connfly* | |
| *X3* | | | *0430450612* | | | | *1* | *Molex* | |
| *X4* | | | *IDC-10MR* | | | | *1* | *Connfly* | |
|  | | |  | | | |  |  | |
| *Подп. И дата* |  | *X5* | | | *HYR-0142AR* | | | | *1* | *New Centress* | |
|  | | |  | | | |  |  | |
| *ZQ1* | | | *Резонатор кварцевый AT-41CD2-8MHz* | | | | *1* | *NDK* | |
|  | | |  | | | |  |  | |
| *Инв. № дубл* |  |  | | | ***Микросхемы*** | | | |  |  | |
| *DA1* | | | *LM1117-3.3* | | | | *1* | *Texas* | |
|  | | |  | | | |  | *Instruments* | |
| *Взам. инв. №* |  | *DA2* | | | *SDM660* | | | | *1* | *Purelogic* | |
|  | | | | | | | | | |
|  |  |  | |  |  | *УРФУ.000000.001 ПЭ3* | | | *Лист* |
|  |  |  | |  |  | *2* |
| *Изм.* | *Лист* | *№ докум.* | | *Подп.* | *Дата* |
|  | | *Поз. обозна-чение* | | | *Наименование* | | | | *Кол.* | *Примечание* | |
| *DA3* | | | *ESP8266-01* | | | | *1* | *Espressif* | |
|  | | |  | | | |  | *Systems* | |
| *DA4* | | | *CH340G* | | | | *1* | *WCH* | |
| *DA5* | | | *LMC7101* | | | | *1* | *Texas* | |
|  | | |  | | | |  | *Instruments* | |
| *DD1* | | | *STM32F103C8T6* | | | | *1* | *STM* | |
|  | | |  | | | |  |  | |
| *K1-K2* | | | *Реле HFD4* | | | | *2* | *Hongfa* | |
|  | | |  | | | |  |  | |
|  | | | ***Выключатели кнопочные*** | | | |  |  | |
| *SB1-SB2* | | | *KLS7-TS3401- 2,5-180* | | | | *2* | *KLS* | |
| *SB3-SB4* | | | *PBS-11A* | | | | *2* | *Jietong Switch* | |
| *SB5* | | | *KLS7-TS3401- 2,5-180* | | | | *1* | *KLS* | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
| *Подп. И дата* |  |  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
|  | | |  | | | |  |  | |
| *Инв. № дубл* |  |  | | |  | | | |  |  | |
| *Взам. инв. №* |  |  | | |  | | | |  |  | |
|  | | | | | | | | | |
|  |  |  | |  |  | *УРФУ.000000.001 ПЭ3* | | | *Лист* |
|  |  |  | |  |  | *3* |
| *Изм.* | *Лист* | *№ докум.* | | *Подп.* | *Дата* |

# ПРИЛОЖЕНИЕ Л

