Оглавление

Техническое задание

Цель разработки: разработка программного обеспечения для учебно-демонстрационного комплекса «Страж» на базе отладочной платы «МилКиТЭС» для управления и мониторинга работы онлайн лаборатории платформ «МилКиСтэнд».

Таблица 1. Технические характеристики

Технические характеристики	
Напряжение питания	+12 В, 3.5 А
Источник питания	Блок питания на *DIN* рейке
Габариты платы	Не более 150x250x90 мм
Управляющий микроконтроллер	К1986*BE92QI*
Количество управляемых линий питания	Не менее 12
Диапазон измеряемого напряжения	0В – 5В
Точность измерения напряжения	± 0.1 В

Классы объекта установки: стационарная аппаратура.

Климатическое исполнение: умеренный (У).

Группа в зависимости от условий эксплуатации: стационарная РЭА, работающая в отапливаемом помещении.

Конструкционные особенности: экспериментальный стенд с блоком питания, автоматом, реле напряжения на *DIN* рейке и отладочной платой «МилКиТЭС».

Пользовательские требования: отображение показаний с линий питания в реальном времени, вывод показаний в онлайн, возможность управления лабораторией дистанционно.

Аннотация

Работа посвящена разработке программного обеспечения для учебно-демонстрационного комплекса «Страж» в составе экспериментального стенда для проверки реализации управления и непрерывных измерений поступающих напряжений с помощью встроенного в микроконтроллер модуля 12-битного аналого-цифрового преобразования (далее АЦП).

Показан процесс написания программного обеспечения:

- получение измерений с АЦП*;*
- обработка полученных измерений*;*
- передача измерений на микроконтроллер «*ESP-12F*» по *UART*;
- отправка показаний с АЦП на сервер;
- принятие запроса с сервера (например, выключить 1 стенд).

Работа содержит 48 страниц, 28 рисунков, 11 таблиц и 10 использованных источников.

Abstract

The work is devoted to the development of software for the educational demonstration complex "Guard" as part of an experimental stand for testing the implementation of control and continuous measurements of incoming voltages using a 12-bit ADC module built into the microcontroller.

The process of writing software is shown:

- *obtaining measurements from the ADC;*
- *processing of the obtained measurements;*
- *transmission of measurements to the "ESP-12F" microcontroller via UART;*
- *sending readings from the ADC to the server;*
- *accepting a request from the server (for example, turn off 1 stand).*

The work contains 48 pages, 28 figures, 11 tables and 10 used sources.

Введение

В современном мире автоматизированный контроль оборудования стал абсолютно необходимым для любой электроники, по причине того, что любая техника нуждается в постоянном контроле. Так, например, в учебно-демонстрационном комплексе «Страж» необходим постоянный контроль печатных плат, чтобы в случае большого ток-потребления или короткого замыкания, система смогла моментально отключить неисправную плату и уведомить пользователя о произошедшей проблеме.

С целью организовать дистанционный контроль оборудования была применена технология «Интернет вещей», благодаря которой система может непрерывно оповещать пользователя о текущем состоянии электронного устройства, а также появляется возможность дистанционного контроля ученого стенда. Благодаря данной технологии повышается мобильность при работе с техникой, а это является одним из главных аспектов в контроле устройства.

Интернет вещей – система сети передачи данных между физическими объектами, оснащенными встроенными средствами и технологиями для взаимодействия друг с другом или внешней средой. Данная технология активно развивается там, где есть потребность в удаленном мониторинге состояния объектов или сборе больших данных с целью последующего анализа.

Задачи*:*

- организовать сбор данных с 12-ти каналов АЦП;
- обработать полученные данные (проверка на аварию, исправность устройства);
- организовать передачу по *UART* между контроллерами «Миландр» и «*ESP*-12*F*» (передача показаний АЦП, аварийные случаи, прием команд от сервера);
- прием команды от сервера*;*
- передача данных на сервер.

Актуальность работы*:*

- данную работу можно считать актуальной благодаря системе автоматизации;
- дистанционный мониторинг показаний в реальном времени;
- система оповещения аварийных случаев и автоматическая обработка аварии;
- контроль оборудования из любой точки мира.

Цель бакалаврской работы – разработка микропрограммы для микроконтроллера К1986ВЕ92QI для обеспечения автоматизации контроля и мониторирования питания стендов инженерного практикума Заочной школы Дубна.

Этапы реализации проекта*:*

1. Изучение технического задания, полученного от руководителя.
2. Сбор необходимой документации.
3. Написание программного обеспечения для микроконтроллера «К1986*ВЕ*92*QI*»:
    a. Написание функции для установки канала мультиплексора;
    b. Инициализация АЦП*;*
    c. Инициализация таймера, установка канала мультиплексора и сбор данных с настраиваемой периодичностью;
    d. Функция анализа снятых показаний, обработка аварии;
    e. Инициализация АЦП: прием данных от *WiFi* модуля, передача данных на сервер;
    f. Вывод показаний на дисплей.
4. Написание программного обеспечения для микроконтроллера (*WiFi* модуль) «*ESP*-12*F*»:
    a. Подключение к *WiFi* сети*;*
    b. Настройка *MQTT* сервера*;*
    c. Настройка *UART* на приемо-передачу данных;
    d. Настройка символьного управления главным контроллером (запрос - ответ).
    e. Обработка полученных данных от сервера;
    f. Обработка полученных данных по *UART*.
5. Отладка проекта.

1. Поиск и выбор аналога протокола передачи данных *IoT*

1.1. Критерии поиска аналогов

Протокол должен соответствовать следующим требованиям:

- наличие встраиваемой библиотеки в среде разработке «*Arduino IDE*»;
- стоимость сервера не более 500 рублей в год;
- наличие шаблона публикация-подписка*;*
- сбор данных от множества узлов.

1.2. Аналоги

1.2.1. Data Distribution Service (DDS)

DDS – реализует шаблон публикации-подписки для отправки и приема данных, событий и команд среди конечных узлов. Узлы-издатели создают информацию, «topic» (темы, разделы: температура, местоположение, давление) и публикуют шаблоны. Узлам, заинтересовавшимся в данных разделах, DDS прозрачно доставляет созданные шаблоны. В качестве транспорта – UDP. Также DDS позволяет управлять параметрами QoS (качество облуживания) [5].

1.2.1.1. Критерии

- Библиотеки для среды разработки «*Arduino IDE*» находятся в закрытом доступе;
- стоимость сервера ~ 1900 р/год*;*
- шаблон публикации-подписки – есть*;*
- сбор данных от множества узлов – есть.

1.2.1.2. Вывод

Рассмотренный протокол не подходит по двум критериям:

- отсутствие библиотек в среде «*Arduino IDE*»;
- стоимость сервера*;*

1.2.2. Constrained Application Protocol (CoAP)

CoAP – с точки зрения пользователя похож на протокол HTTP, но отличается малым размером заголовков, что подходит для сетей с ограниченными возможностями. Использует архитектуру клиент-сервер и подходит для передачи информации о состоянии узла на сервер (сообщения GET, PUT, HEAD, POST, DELETE, CONNECT). В качестве транспорта – UDP [5].

1.2.2.1. Критерии

- Наличие встраиваемых библиотек в среде «*Arduino IDE*» – нет;
- стоимость сервера ~ 900 р/год*;*
- шаблон публикация-подписка – нет*;*
- сбор данных от множества узлов – есть.

1.2.2.2. Вывод

Рассмотренный протокол не подходит по трем критериям:

- наличие встраиваемых библиотек в среде «*Arduino IDE*»;
- стоимость сервера*;*
- отсутствует шаблон публикация-подписка.

1.2.3. Extensible Messaging and Presence Protocol (XMPP)

XMPP – открытый, основанный на *XML*, свободный для использования протокол для мгновенного обмена сообщениями и информацией о присутствии в режиме, близком к режиму реального времени [5].

1.2.3.1. Критерии

- Наличие встраиваемых библиотек в среде «*Arduino IDE*» – есть;
- стоимость сервера – 3000 р./год*;*
- шаблон публикация-подписка – есть*;*
- сбор данных от множества узлов – есть.

1.2.3.2. Вывод

Рассмотренный протокол не подходит по причине высокой стоимости сервера.

1.2.4. Message Queue Telemetry Transport (MQTT)

MQTT – осуществляет сбор данных от множества узлов и передачу на сервер. Основывается на модели издатель-подписчик с использованием промежуточного сервера – брокера (приоритезация сообщений, формирование очередей и др.). В качестве транспорта – TCP [5].

1.2.4.1. Критерии

- Наличие встраиваемых библиотек в среде «*Arduino IDE*» – есть;
- стоимость сервера ~ 300 р./год;
- *шаблон публикация-подписка* – есть;
- сбор данных от множества узлов – есть.

1.2.4.2. Вывод

Рассмотренный протокол отлично подходит для реализации беспроводной передачи данных между устройствами.

1.3. Вывод

Среди рассмотренных аналогов был выбран *MQTT* протокол, т.к. только он удовлетворял всем критериям.

2. Разработка программного обеспечения для микроконтроллера «К1986*BE*92*QI*»

2.1. Установка канала мультиплексора

2.1.1. Принцип работы мультиплексора

Мультиплексор – это комбинационное устройство, предназначенное для коммутации сигналов и потоков данных в линиях связи по заданным адресам и маршрутам. Основное назначение мультиплексоров заключается в передачи сигналов из нескольких входов на один выход, причем выбор выхода может осуществляться при помощи сочетания определенных управляющих сигналов.

Для реализации мультиплексора необходимо создать дешифратор, имеющий 2 входных сигнала и четыре выходных (см. рис. 2.1.1).

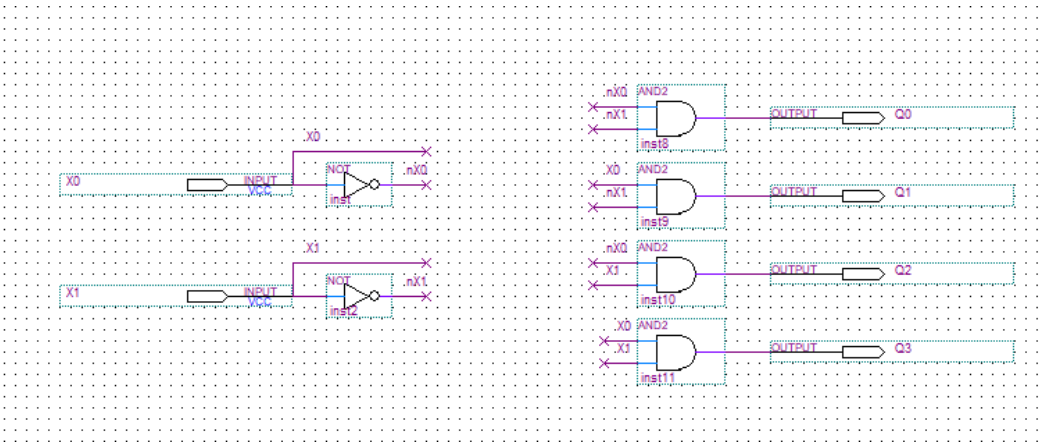


Рис. 2.1.1. Схематическое представление дешифратора

Из результатов моделирования (см. рис. 2.1.2) видно, что данная схема работает в точности по таблице истинности дешифратора (см. таблицу 2).

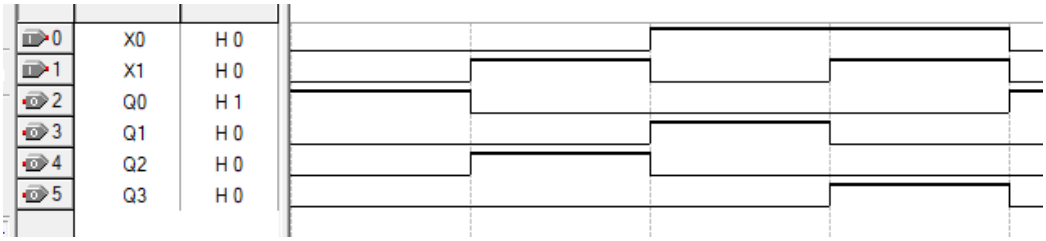


Рис. 2.1.2. Моделирование дешифратора

Таблица 2.1.1. Таблица истинности Дешифратора

X_1	X_0	Q_0	Q_1	Q_2	Q_3
0	*0*	*1*	*0*	*0*	*0*
0	*1*	*0*	*1*	*0*	*0*
1	*0*	*0*	*0*	*1*	*0*
1	*1*	*0*	*0*	*0*	*1*

На основе спроектированного дешифратора был спроектирован мультиплексор (см. рис. 2.1.3).

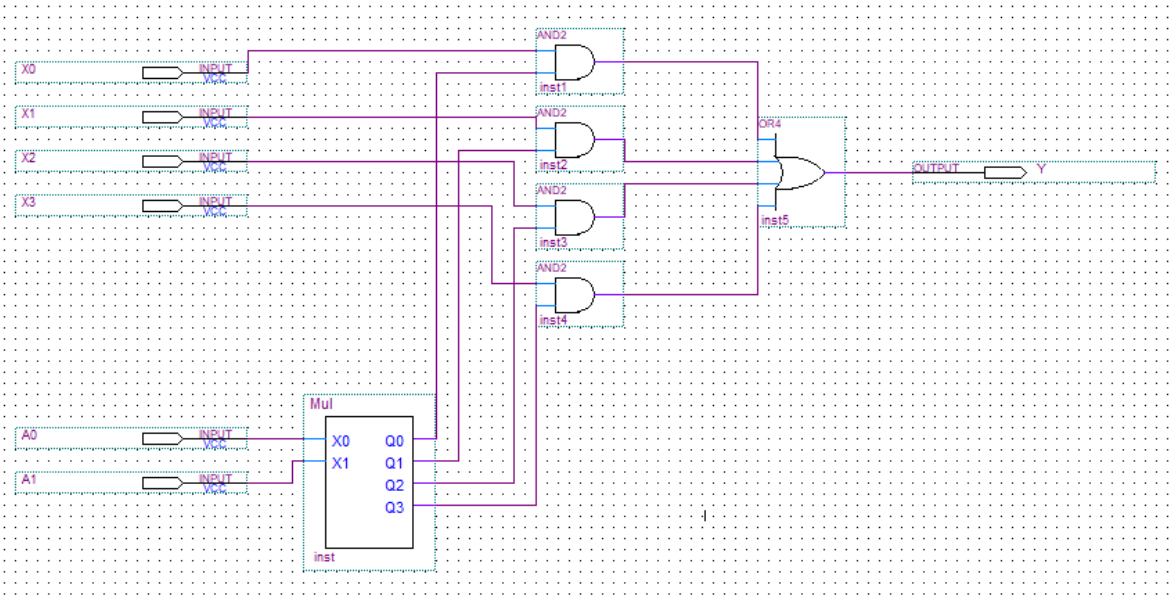


Рис. 2.1.3. Схематическое представление Мультиплексора

Из результатов моделирования (см. рис. 2.1.4) можно сделать вывод, что принцип работы мультиплексора отлично подходит для переключения каналов между стендами, с последующим снятием показаний АЦП (см. таблицу 2.1.2).

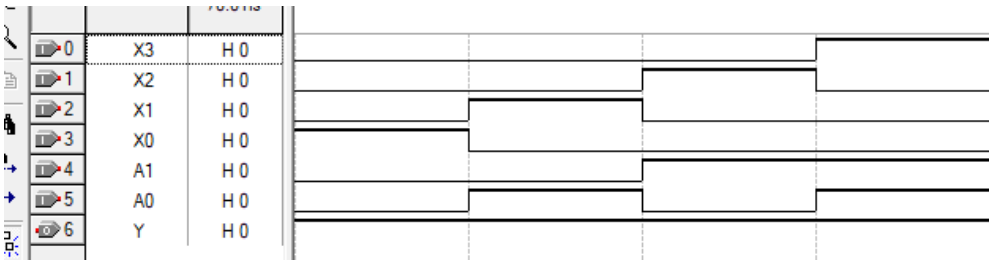


Рис. 2.1.4. Моделирование мультиплексора

Таблица 2.1.2. Мультиплексор

X_3	X_2	X_1	X_0	A_1	A_0	Y
0	*0*	*0*	*1*	*0*	*0*	*1*
0	*0*	*1*	*0*	*0*	*1*	*1*
0	*1*	*0*	*0*	*1*	*0*	*1*
1	*0*	*0*	*0*	*1*	*1*	*1*

2.1.2. Назначение портов

Порты ввода-вывода общего назначения – это основной интерфейс связи микроконтроллера с внешними устройствами.

Порты состоят из параллельных линий, каждая из которых может использоваться для ввода или вывода информации. Порты могут быть ориентированы как на цифровые, так и на аналоговые сигналы. Наиболее часто используют цифровой режим портов; аналоговый задействуют лишь при работе с аналого-цифровыми и цифро-аналоговыми преобразователями, компараторами и внешними кварцевыми резонаторами.

Линии, сконфигурированные как цифровой ввод, используют для подключения к системе каких-либо коммутаторов (кнопок, электромагнитных реле) или приема цифрового сигнала. Иными словами, для ввода внешних сигналов в микроконтроллер.

Линии, сконфигурированные как цифровой вывод, используются для формирования управляющих воздействий. Это позволяет, например, зажечь светодиод, или лампу накаливания, запустить электродвигатель, передать логическую последовательность. Иными словами, такие линии предназначены для вывода сигналов на внешние устройства.

Почти все линии микроконтроллеров серии 1986BE9x могут функционировать в режиме цифрового ввода или вывода, но не ограничены этим. Функция каждой линии задаётся программно. Для удобства использования линии ввода-вывода на аппаратном уровне объединены в порты. Порты именуются латинскими литерами:

- PORTA (порт A)*;*
- PORTB (порт B);
- PORTC (порт C) и т.д.

Линии при этом нередко называют так, что, например, нулевая линия порта C – это PC0, а пятая линия порта F – PF5, и т.д.

Схема расположения выводов микроконтроллера К1986ВЕ92QI (корпус LQFP64) показана на рисунке 2.1.5.

Функциональная схема отдельной линии ввода-вывода изображена на рисунке 2.1.6. Все элементы схемы, за исключением контактного, расположены внутри микроконтроллера.

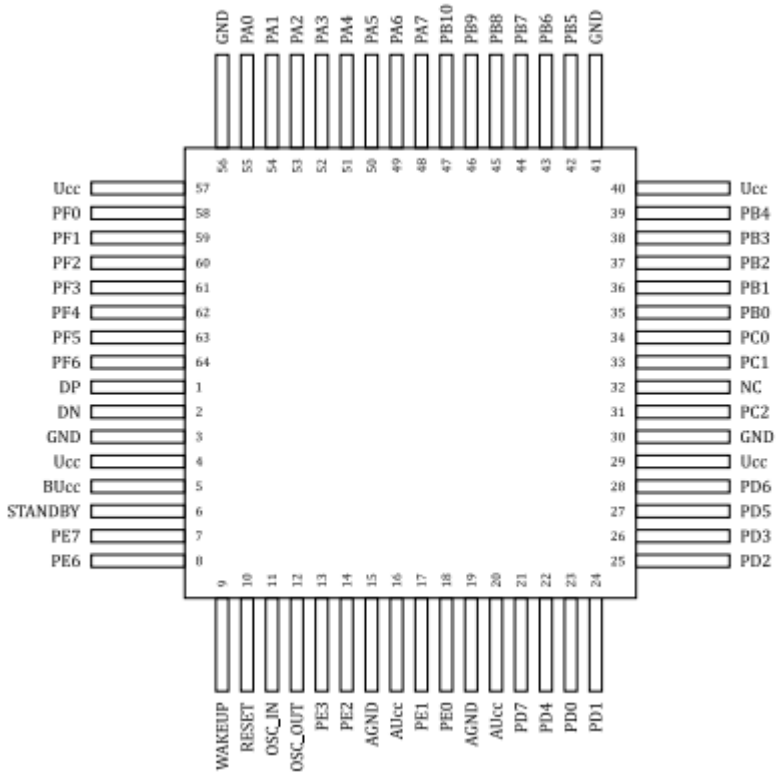


Рис. 2.1.5. Схема расположения выводов микроконтроллера К1986ВЕ92QI

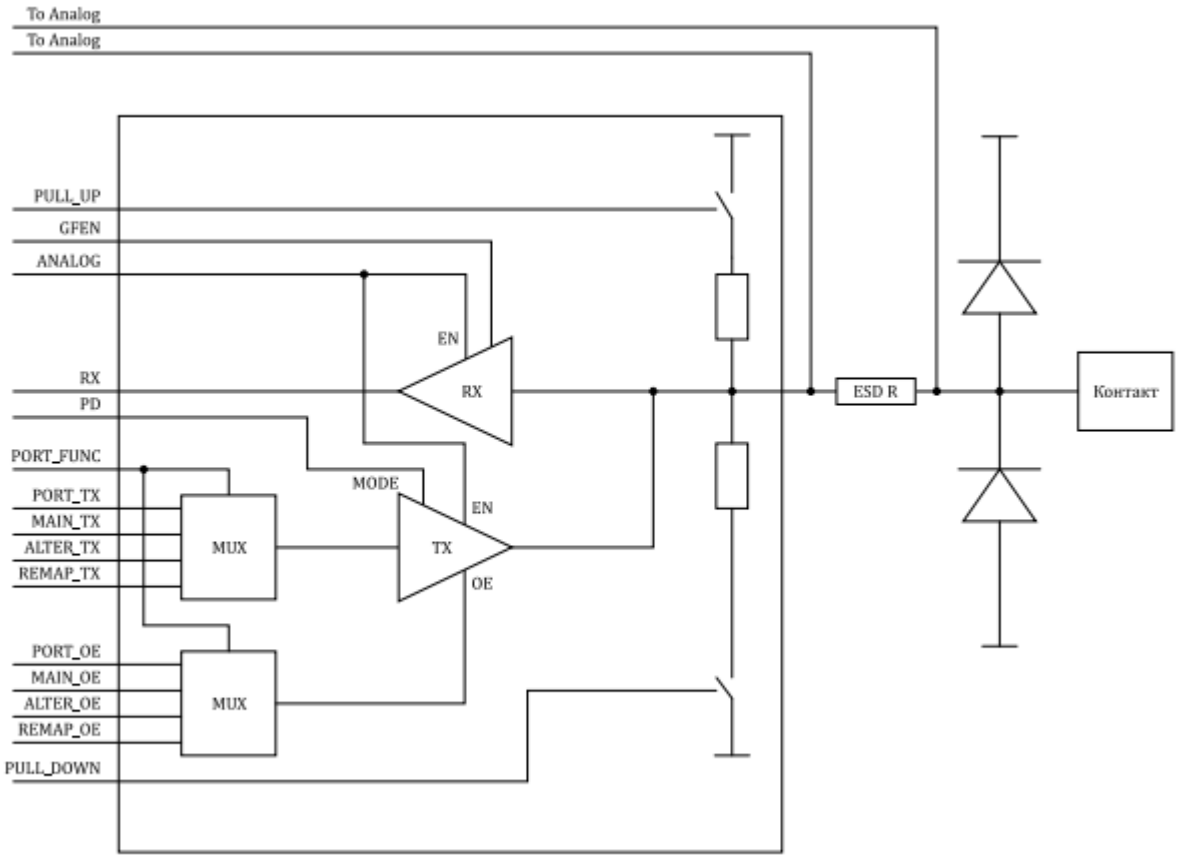


Рис. 2.1.6. Функциональная схема отдельной линии ввода-вывода

Таблица 2.1.3. Распределение линий ввода-вывода по портам в микроконтроллерах К1986ВЕ92QI

Наименование порта	**Количество линий**	**Наименование линий**
PORTA	*8*	*PA0…PA7*
PORTB	*11*	*PB0…PB10*
PORTC	*3*	*PC0…PC2*
PORTD	*8*	*PD0…PD7*
PORTE	*6*	*PE0…PE3, PE6…PE7*
PORTF	*7*	*PF0…PF6*

На цифровые вводы микроконтроллера можно подавать напряжение до 5 В; однако для вводов, соединенных с аналоговыми блоками (порты D и E), максимальное напряжение составляет 3.3 В.

Два быстродействующих диода (один из которых подключается к цепи питания, другой – к земле) предназначены для ограничения входного напряжения; резистор ESD R (англ. Electric Static Discharge) – для ограничения входного тока. Если напряжение на вводе окажется выше питания, то верхний диод откроется, и оно будет стравлено на фильтры источника. Если же напряжение окажется отрицательным, то оно будет нейтрализовано через нижний диод, на землю. Такое схемотехническое решение позволяет повысить долговечность микросхемы. Впрочем, оно защищает лишь от слабых импульсов и помех; если по ошибке подать на ввод 6-7 В, то никакие диоды его не спасут.

Цифровые выводы микроконтроллеров серии 1986BE9x способны выдавать ток до ~6 мА каждый (до ~100 мА суммарно для каждого порта). Это позволяет, например, подключить к выводам яркие светодиоды. Однако для управления элементами с большей нагрузкой (лампы, приводы, двигатели) необходимо использовать дополнительные электрические каскады.

2.1.3. Инициализация портов

Порты ввода-вывода в микроконтроллерах серии 1986BE9x имеют широкий функционал, поэтому перед использованием в каждом конкретном алгоритме должны быть соответствующим образом настроены (см. приложение 5).

Для программного управления линиями каждый порт содержит 8 регистров (таблица 2.1.4).

Все блоки микроконтроллера (в том числе порты ввода-вывода) по умолчанию не тактируются в целях снижения энергопотребления; поэтому перед началом работы необходимо включить тактирование целевого порта с помощью регистра «*PER_CLOCK*».

Таблица 2.1.4. Регистры управления портами ввода-вывода

Адресное смещение	Условное наименование	Описание
0x00	*RXTX*	Состояние
0x04	*OE*	Направление передачи
0x08	*FUNC*	Функция
0x0C	*ANALOG*	Режим работы
0x10	*PULL*	Подтяжка
0x14	*PD*	Управление выводом
0x18	*PWR*	Крутизна фронтов
0x1C	*GFEN*	Цифровой фильтр

2.1.3.1. Регистр *RXTX*

Функциональное имя битов: *RXTX*[15..0].

Состояния линии порта*:*

- *0* – низкий логический уровень (0 В)*;*
- *1* – высокий логический уровень (3.3 В).

Регистр RXTX позволяет управлять всеми линиями порта. Каждый бит регистра отражает состояние соответствующей линии: 0 – низкий логический уровень, 1 – высокий. Работу с регистром следует выполнять уже после того, как целевые линии порта будут настроены.

Если линия выполняет функцию цифрового вывода, то запись значения в регистр позволяет управлять её состоянием. К примеру, для создания напряжения на линии PA2 можно написать так:

$$\text{MDR\_PORTA}{-}{>}\ \text{RXTX} \mid = \ (1\ <<\ 2);$$

Если линия работает как цифровой ввод, то чтение регистра позволяет определить логический уровень внешнего сигнала. Например, сигнал с линии PA3 можно сохранить в переменную таким образом:

$$\text{int data}\ =\ (\text{MDR\_PORTA}{-}{>}\ \text{RXTX}\ >>\ 3)\ \&\ 0\text{x}01;$$

Сдвиг вправо на 3 позиции позволяет перенести целый бит в младший разряд, а операция «&» со значением 0*x*01 маскирует старшие биты.

2.1.3.2. Регистр *OE*

Функциональное имя битов: *OE*[15..0].

Направление линии:

- 0 – ввод данных*;*
- 1 – вывод данных.

Регистр OE (Output Enable) разрешает или запрещает программное управление линиями; или, иными словами, определяет направление передачи данных. Чтобы, к примеру, использовать линию PA2 для вывода данных, а линию PA3 – для ввода, следует выполнить такую настройку:

$$\text{MDR\_PORTA} \rightarrow \text{OE} \mid = \ (1\ \ll\ 2);$$

$$\text{MDR\_PORTA} -> \text{OE} \,\&= \sim(1 << 3);$$

2.1.3.3. Регистр *FUNC*

Функциональное имя битов:

- *FUNCx[31..2] :*
  - Аналогично *FUNC0* для остальных линий
- *FUNC0[1..0]*
  - 00 – ввод/вывод общего назначения*;*
  - 01 – основная*;*
  - 10 – альтернативная*;*
  - 11 – переопределенная.

Регистр FUNC задает функцию линии ввода-вывода. Дело в том, что каждая линия микроконтроллеров может быть не только цифровым вводом или выводом, но и выполнять функции встроенных периферийных блоков. Например, линия PD0 может быть использована в качестве инверсного канала таймера 1, прямого канала таймера 3, линии приёма контроллера UART2. Данные функции условно именуют основной, альтернативной, переопределенной. Такой механизм необходим в связи с большим количеством периферии микроконтроллера и ограниченным количеством выводов.

Для выбора функции каждой линии в регистре предусмотрено 2 бита. В частности, для линии PA0 – это биты 0 и 1, для линии PA1 – биты 2 и 3, и т.д. Таким образом, чтобы сконфигурировать линию PA3 как ввод-вывод общего назначения (00_2) можно написать так:

$$\text{MDR\_PORTA} -> \text{FUNC} \,\&= \sim(3 << 6);$$

Для каждой линии предназначено два бита. Целевой является линия 3, поэтому выполняется сдвиг влево на 6 бито. Значение 3 используется, чтобы сбросить сразу два бита.

Если требуется задать основную или альтернативную функцию (значение 01_2 и 10_2 соответственно), то это следует делать в две операции: сначала сбросить оба бита в ноль, а затем установить требуемый бит в единицу.

Если ограничиться только установкой битов, то не удастся гарантировать верное значение этого параметра.

2.1.3.4. Регистр *ANALOG*

Функциональное имя битов: *ANALOG*[15..0].

Режим работы линии:

- *0* – аналоговый*;*
- 1 – цифровой.

Регистр *ANALOG* определяет режим работы линии: аналоговый или цифровой. В большинстве случаев используется цифровой режим; аналоговый режим выбирают лишь при работе с аналого-цифровым, цифро-аналоговым преобразователями или компаратором.

Для примера, сделать линию PA3 цифровой можно так:

$$\text{MDR\_PORTA} -> \text{ANALOG} \mid = (1 << 3);$$

2.1.3.5. Регистр *PULL*

Функциональное имя битов:

- *PULL_UP*[31..16] – подтяжка линии к потенциальному питанию (3.3 В)*:*
  - 0 – подтяжка отключена*;*
  - 1 – подтяжка включена.
- *PULL_DOWN*[15..0] – подтяжка линии к потенциалу земли (0 В):
  - *0* – подтяжка отключена*;*
  - 1 – подтяжка включена.

Регистр *PULL* управляет подключением линий к земле либо цепи питания через специальный подтягивающий резистор номиналом ~50 КОм.

В целом, подтяжка повсеместно используется в электрических схемах для гарантии устойчивого состояния на линии в случаях, когда управляющий элемент отключен или его выводы находятся в высокоимпедансном состоянии.

Идейно, наличие таких резисторов непосредственно внутри микроконтроллера позволяет снизить размер печатной платы разрабатываемого устройства, количество требуемых электронных компонентов, число паек и стоимость изделия в целом.

Однако с учетом относительно небольшого вырабатываемого тока цифровых выводов и высокого значения сопротивления этих резисторов, им трудно найти практическое применение, и в большинстве случаев их отключают, а для подтяжки задействуют внешние элементы.

Структура регистра *PULL* разбита на два блока: младшие 16 битов отвечают за подтяжку линий к потенциалу земли, старшие 16 битов – за подтяжку к потенциалу питания. В этом случае для отключения подтяжек линии *PA*3 можно записать так:

$$\text{MDR\_PORTA}{-}{>}\ \text{PULL}\ \&=\ {\sim}(1\ {<}{<}\ 3);$$

$$\text{MDR\_PORTA}{-}{>}\ \text{PULL}\ \&=\ {\sim}(1\ {<}{<}\ 19);$$

2.1.3.6. Регистр *PD*

Функциональное имя битов:

- *SHM[31..16]* – Триггер Шмитта*:*
  - *0* – отключен*;*
  - *1* – включен*;*
- *PD[15..0]* – управление линией*:*
  - *0* – драйвер*;*
  - *1* – открытый коллектор.

Регистр PD (Port Driver) разделён на два функциональных блока.

Одноименный блок PD занимает 16 младших битов и определяет режим работы цифрового вывода. Реализовано два режима: «драйвер» и «открытый коллектор». Следует отметить, что этот параметр не используется для линий, настроенных на ввод данных.

Режим «драйвер» (или его ещё иногда называют «push-pull») является стандартным режимом работы цифрового вывода. В этом режиме линия подключается через два транзистора, один из которых соединен с землей, другой – с питанием (см. рис. 2.1.7). В каждый момент времени открыт только один транзистор, что позволяет создать два устойчивых состояния – низкое и высокое.

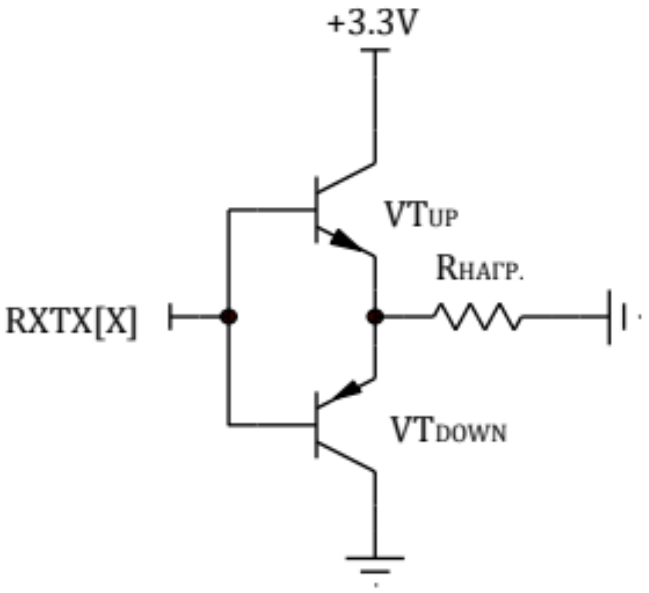


Рис. 2.1.7. Схема подключения цифрового вывода в режиме «драйвер»

В режиме «открытый коллектор» линия подключается к базе одного транзистора, коллектор которого соединен с землей, а эмиттер открыт для соединения с внешней схемой. Таким образом, вывод может находиться в двух состояниях: низком, если транзистор открыт; и высокоимпедансном (англ. Hi-Z), если транзистор закрыт. В высокоимпедансном состоянии уровень сигнала на линии не соответствует ни логическому нулю, ни логической единице, поэтому данное состояние еще называют третьим. По этой причине данный режим обычно используют для работы в схемах с подтяжкой к питанию (см. рис. 2.1.8).

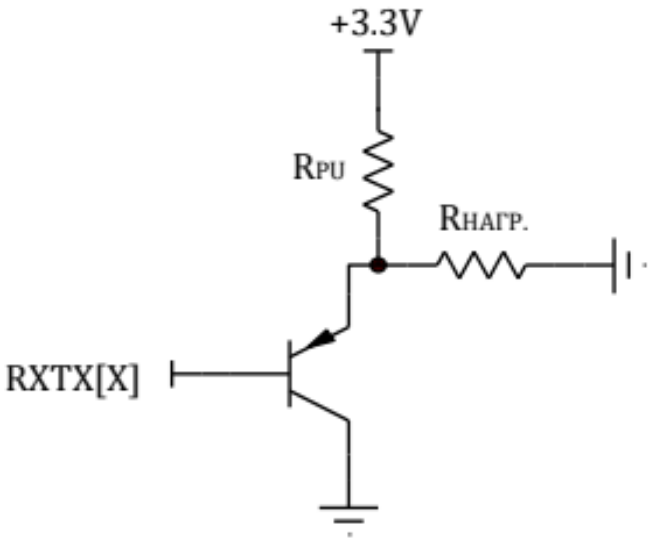


Рис. 2.1.8. Схема подключения цифрового вывода в режиме «открытый коллектор» с подтяжкой к питанию

Блок SHM занимает 16 старших битов и позволяет подключить к цифровому вводу триггер Шмитта. Данный триггер идейно расширяет петлю гистерезиса на линии. Но практическая значимость этого параметра в микроконтроллерах серии 1986BE9x довольно мала.

2.1.3.7. Регистр *PWR*

- *PWRx*[31..2] – аналогично *PWR*0 для остальных линий;
- *PWR*0[1..0] – крутизна фронтов выходных импульсов:
  - *00* – передача отключена*;*
  - 01 – низкая крутизна (длительность фронта ~100 нс);
  - 10 – средняя крутизна (длительность фронта ~20 нс);
  - 11 – высокая крутизна (длительность фронта ~10 нс).

Регистр *PWR* (*Power*) определяет крутизну фронтов выходных импульсов цифрового вывода. Прямоугольные импульсы, широко используемые в цифровой электронике, в действительности всегда являются трапецеидальными. То, насколько эти импульсы близки к прямоугольной форме, определяется отношением их длительностью ко времени переходного процесса: чем меньше время переходного процесса, тем более крутыми становятся фронты импульсов.

На рисунке 2.1.9 приведена сравнительная осциллограмма прямоугольных импульсов на частоте 2.5 МГц с низкой и высокой крутизной фронтов. Как видно из рисунка, высокая крутизна фронтов обеспечивает большую длительность сигнала («полка»). При этом, однако, возникает колебательность переходного процесса, создающая некоторый спектр помех. В целом, высокая крутизна импульсов необходима при работе на высоких частотах (более ~5 МГц); например, при программной реализации высокоскоростного интерфейса. В остальных же случаях правильнее будет использовать низкую крутизну.

Рис. 2.1.9. Осциллограмма прямоугольных импульсов: сверху – низкая крутизна фронтов, снизу – высокая крутизна фронтов

Для определения мощности передачи для каждой линии в структуре регистра PWR отведено 2 бита. По аналогии с регистром FUNC, для задания, к примеру, низкой крутизны фронтов выходных импульсов на линии PA2 следует указать:

$$\text{MDR\_PORTA}{-}{>}\ \text{PWR}\ \&=\ {\sim}(3\ {<}{<}\ 4);$$

$$\text{MDR\_PORTA}{-}{>}\ \text{PWR}\ |=\ (1\ {<}{<}\ 4);$$

2.1.3.8. Регистр *GFEN*

- *GFEN[15..0]* – функциональное имя битов*;*
- Цифровой фильтр*:*
  - 0 – отключен*;*
  - *1* – включен.

Регистр *GFEN* (*Grid Filter Enable*) позволяет подключить к цифровому вводу фильтр, который будет отсеивать входные импульсы длительностью менее 10 нс. Т.е. если на ввод с таким фильтром попадет указанный импульс, то он будет интерпретирован, как помеха, и не повлияет на логическое состояние линии.

Цифровой фильтр значительно увеличивает помехозащищенность приёмного тракта, однако его не следует использовать при работе с высокочастотными сигналами (более ~25 МГц), т.к. это может привести к потере информации.

Чтобы включить фильтрацию на линии PA3 достаточно написать:

$$\mathrm{MDR\_PORTA}{-}{>}\ \mathrm{GFEN}\ |{=}\ (1\ {<}{<}\ 3);$$

2.1.3.9. Инициализация мультиплексора

На основе документации мультиплексора «*ADG*726» (см. таблицу 2.1.5) и технического задания (см. рис. 2.1.10), было необходимо проинициализировать следующие пины:

- *PA*0-*PA*7,*PB*8-*PB*10,*PF*5 — на выход, управление реле с 1 по 12 плату;
- *PC*0-*PC*2,*PF*4 — на выход, установки канала мультиплексора;
- *PD*3 — в режиме АЦП, измерение напряжения на установленном канале;
- *PD*5 — АЦП, измерение общего напряжения;
- *PD*2 — на выход, вкл./выкл. мультиплексора.

Таблица 2.1.5. Таблица истинности мультиплексора «*ADG*726»

A3	***A2***	***A1***	***A0***	$\overline{\mathbf{EN}}$	$\overline{\mathbf{CSA}}$	$\overline{\mathbf{CSB}}$	$\overline{\mathbf{WR}}$	**Выход**
×	×	×	×	×	*1*	*1*	L → H	Защелки управляют входными данными
×	×	×	×	×	*1*	*1*	×	Нет изменений в состоянии переключателя
×	×	×	×	*1*	×	×	×	Ничего
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*S1A* в *DA, S1B* в *DB*
0	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*S2A* в *DA, S2B* в *DB*
0	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*S3A* в *DA, S3B* в *DB*
0	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*S4A* в *DA, S4B* в *DB*
0	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*S5A* в *DA, S5B* в *DB*
0	*1*	*0*	*1*	*0*	*0*	*0*	*0*	*S6A* в *DA, S6B* в *DB*
0	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*S7A* в *DA, S7B* в *DB*
0	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*S8A* в *DA, S8B* в *DB*
1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*S9A* в *DA, S9B* в *DB*
1	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*S10A* в *DA, S10B* в *DB*
1	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*S11A* в *DA, S11B* в *DB*
1	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*S12A* в *DA, S12B* в *DB*

Рис. 2.1.10. Схематическое подключение мультиплексора к МилКиТЭС

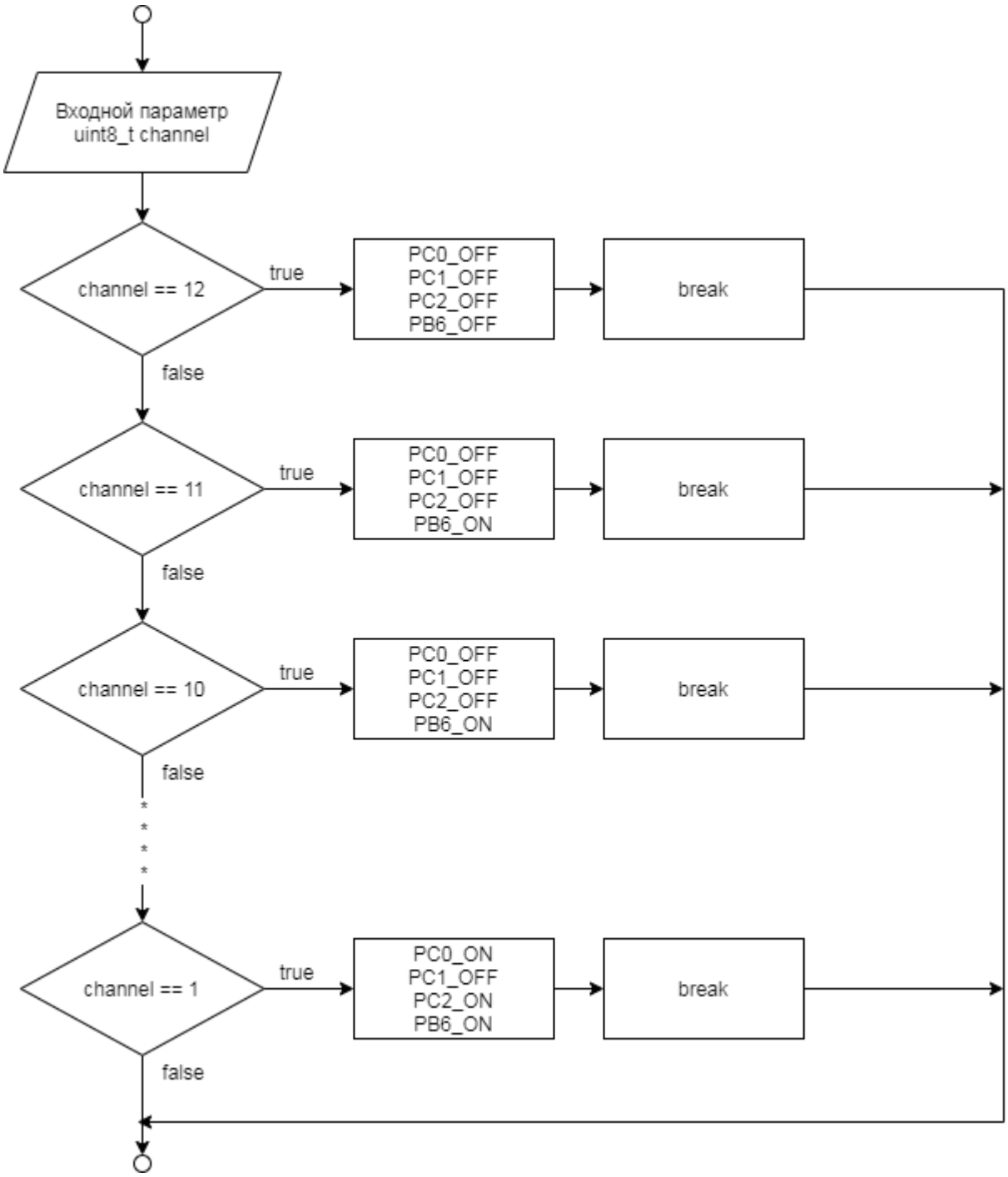
Руководствуясь документацией на микроконтроллер «К1986ВЕ92QI» настройка регистров была произведена следующем образом (см. приложение 4):

- *OE* (направление порта) – выход;
- *FUNC* (режим работы порта) – ввод/вывод;
- *ANALOG* (аналоговый режим работы порта) – цифровой;
- *PULL* (подтяжка порта) – отключена;
- *PD* (режим работы выходного драйвера) – управление выводом;
- *PWR* (режим мощности передачи) – высокая;
- *GFEN* (режим работы входного фильтра) – не используется.

Таким образом, для работы с учебно-демонстрационным комплексом «Страж» были назначены следующие директивы:

- *MX_EN/MX_DIS* – вкл./выкл. мультиплексора;
- *MX_CSa_EN*/*MX_CSa_DIS* – разрешение/запрет каналов;
- *MX_CSb_EN*/*MX_CSb_DIS* – разрешение/запрет каналов;
- *S1_EN, S2_EN…S12_EN* – вкл. с 1 по 12 стенд;
- *S1_DIS, S2_DIS…S12_DIS* – выкл. с *1* по *12* стенд.

Также для дальнейшей работы с мультиплексором была написала функция установки канала мультиплексора (см. блок-схему 2.1.1), во входной параметр передается номер канала (стенда).



Блок-схема 2.1.1. Установка канала мультиплексора

2.2. Инициализация АЦП

2.2.1. Теория

В микроконтроллере реализовано два 12-разрядных АЦП. С помощью АЦП можно оцифровать сигнал от 16 внешних аналоговых выводов порта D и от двух внутренних

каналов, на которые выводятся датчик температуры и источник опорного напряжения Скорость выборки составляет до 512 тысяч преобразований в секунду для каждого АЦП.

В качестве опорного напряжения преобразования могут выступать:

- питание АЦП с выводов *AUCC* и *AGND*;
- внешние сигналы с выводов *ADC*0*_REF+* и *ADC_REF-*.

Контроллер АЦП позволяет*:*

- оцифровать один из 16 внешних каналов;
- оцифровать значение встроенного датчика температуры;
- оцифровать значение внутреннего источника опорного напряжения;
- осуществлять автоматический опрос заданных каналов;
- выработать прерывание при выходе оцифрованного значения за заданные пределы;
- запускать два АЦП синхронно для увеличения скорости выборки.

Для осуществления преобразования требуется 28 тактов синхронизации *C_ADC*. В качестве синхросигнала может выступать частота процессора либо частота, формируемая в блоке «Сигналы тактовой частоты». Выбор частоты осуществляется с помощью бита *Cfg_REG_CLKS*. Для получения частоты *PCLKd* в контроллере АЦП частота *PCLK* может быть поделена с помощью битов *Cfg_REG_DIVCLK*[3:0].

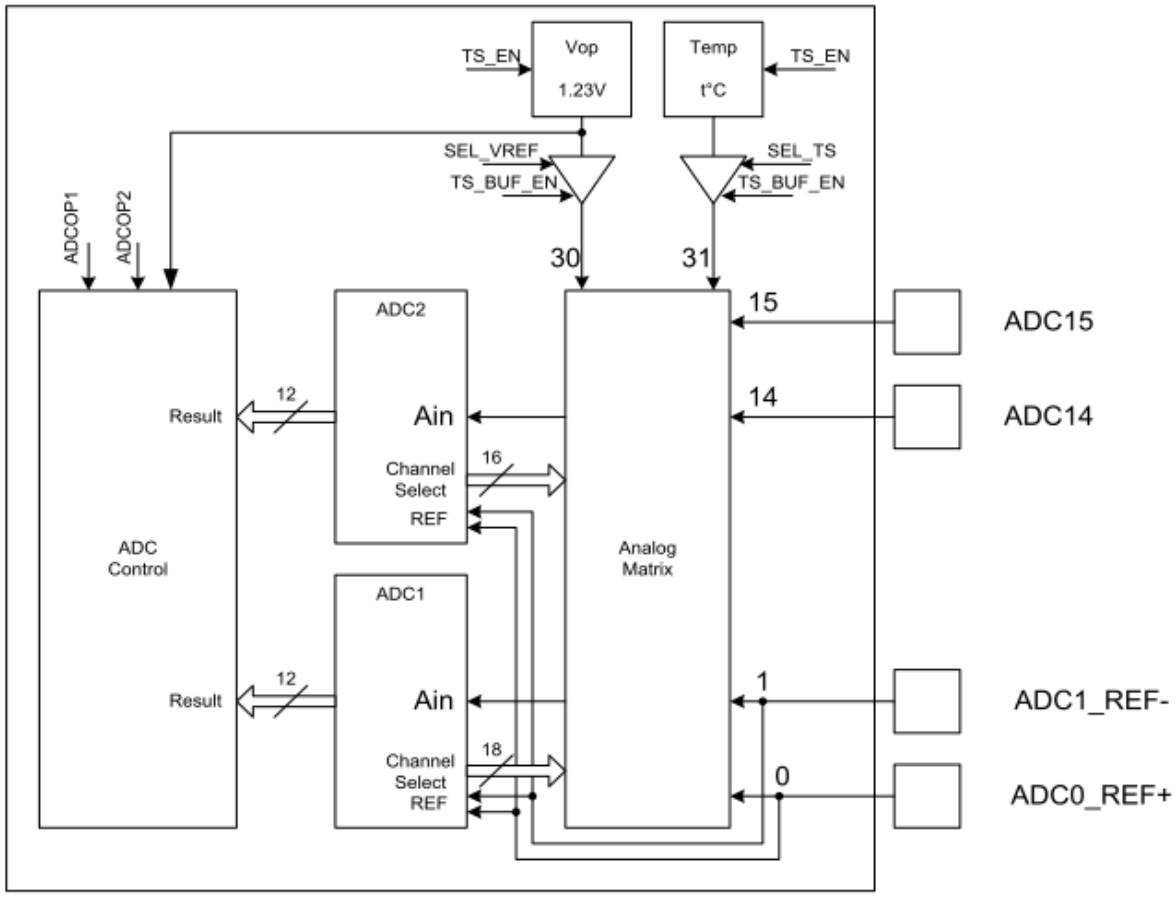


Рис. 2.2.1. Структурная схема контроллера АЦП

Для включения АЦП необходимо установить бит *Cfg_REG_ADON*. Для снижения тока потребления вместо собственного источника опорного напряжения в АЦП может использоваться источник опорного напряжения датчика температуры. Для этого необходимо включить блок датчика температуры и источник опорного напряжения, установив бит *TS_EN* в 1. После включения можно использовать источник опорного напряжения для первого и второго АЦП вместо их собственных. Для этого необходимо установить биты *ADCx_OP* в единицу. Для преобразования необходимо, чтобы выводы, используемы АЦП у порта *D*, были сконфигурированы как аналоговые и были отключены какие-либо внутренние подтяжки.

2.2.2. Описание регистров блока контроллера АЦП

Таблица 2.2.1. Описание регистров блока контроллера АЦП

Базовый Адрес	**Название**	**Описание**
0*x*4008_8000	*MDR_ADC*	Контроллер *ADC*
Смещение		
0x00	*MDR_ADC->ADC1_CFG*	Регистр управления *ADC1*
0x04	*MDR_ADC->ADC2_CFG*	Регистр управления *ADC2*

0x08	*ADC1_H_LEVEL*	Регистр верхней границы *ADC1*
0x0C	*ADC2_H_LEVEL*	Регистр верхней границы *ADC2*
0x10	*ADC1_L_LEVEL*	Регистр нижней границы *ADC1*
0x14	*ADC2_L_LEVEL*	Регистр нижней границы *ADC2*
0x18	*ADC1_RESULT*	Регистр результата *ADC1*
0x1C	*ADC2_RESULT*	Регистр результата *ADC2*
0x20	*ADC1_ STATUS*	Регистр статуса *ADC1*
0x24	*ADC2_ STATUS*	Регистр статуса *ADC2*
0x28	*ADC1_CHSEL*	Регистр выбора каналов перебора *ADC1*
0x2C	*ADC2_CHSEL*	Регистр выбора каналов перебора *ADC2*

Таблица 2.2.2. Описание бит регистра *ADC1_CFG*

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…28	*Delay ADC [3:0]*	Задержка между началом преобразования *ADC1* и *ADC2* при последовательном переборе либо работе на один канал*:* 0000 – 1 такт*;* 0001 – 2 такта*;* … 1111 – 16 тактов
27…25	*Delay Go [2:0]*	Дополнительная задержка перед началом преобразования после выбора канала*:* *000* – 1 такт 001 – 2 такта … 111 – 8 тактов
24…21	*TR [3:0]*	Подстройка опорного напряжения
20	*SEL VREF*	Выбор для оцифровки внутреннего источника опорного напряжения 1,23 В*:* 0 – не выбран*;* 1 – выбран
19	*SEL TS*	Выбор оцифровки датчика температуры*:* 0 – не выбран 1 – выбран
18	*TS BUF EN*	Включения выходного усилителя для датчика температуры и источника опорного напряжения*:* *0* – выключен*;* 1 – включен
17	*TS EN*	Включение датчика температуры и источника опорного напряжения*:*

		0 – выключен*;* 1 – включен
16	*Cfg* *Sync* *Conver*	Запускает работу двух АЦП одновременно
15…12	*Cfg* *REG* *DIVCLK*	Выбор коэффициента деления частоты процессора
11	*Cfg* *M_REF*	Выбора источника опорных напряжений*:* 0 – внутреннее опорное напряжение*;* 1 – внешнее опорное напряжение
10	*Cfg* *REG* *RNGC*	Разрешение автоматического контроля уровней*:* *0* – не разрешена*;* 1 – разрешена выборка флага при выходе за диапазон в регистрах границы
9	*Cfg* *REG* *CHCH*	Выбор переключения каналов*:* *0* – используется только выбранный канал*;* 1 – переключение включено (перебиваются каналы, выбранные в регистре выбора канала)
8…4	*Cfg* *REG* *CHS* *[4:0]*	Выбор аналогового канала, по которому поступает сигнал для преобразования*:* *00000 – 0* канал*;* *00001 – 1 канал;* … 11111 – 31 канал
3	*Cfg* *REG* *SAMPLE*	Выбор способа запуска АЦП: 0 – одиночное; 1 – последовательное. Автоматический запуск после завершения предыдущего преобразования
2	*Cfg* *REG* *CLKS*	Выбор источника синхросигнала *C_ADC* работы *ADC*: *0 – PCLKd* 1 – *ADC_CLK*
1	*Cfg* *REG* *GO*	Начало преобразования. Запись «1» начинает процесс преобразования, сбрасывается автоматически
0	*Cfg* *REG* *ADON*	Включение АЦП*:* 0 – выкл. 1 – вкл.

2.2.3. Настройка АЦП

Для работы с АЦП необходимо включить тактирования всего модуля АЦП в регистре управления тактовой частотой периферийных блоков.

Также требуется задать конфигурацию, основываясь на таблице 2.2.2*:*

- Работа АЦП (*REG ADON*) – вкл.
- Длительность тактовой частоты (*REG DIVCLK*) – 8 тактов*;*
- Дополнительная задержка при выборе канала (*DELAY GO*) – 8 тактов ядра*;*

Также следует настроить порты ввода/вывода на вход в режим работы АЦП*:*

Таблица 2.2.3. Настройка портов АЦП

Канал АЦП	**Направление передачи (*OE*)**	**Режим работы (*ANALOG*)**
PD3 (мультиплексор)	Вход	Аналоговый режим
PD5 (плата «*master*»)		
PD5 (общее напряжение)		

2.2.3. Функции АЦП

Для работы с АЦП коррелируют три вспомогательные функции (см. приложение 6)*:*

- Начало преобразования (*MCU_ADC_start_conv*) – запись логической единицы в регистр преобразования *(REG GO,* см. таблицу 2.2.2*);*
- Установка канала АЦП *(MCU_ADC_set_ch)* – запись номера канала в регистр «*REG CHS*»*;*
- Чтение результата преобразования *(MCU_ADC_read)* – возвращает результат преобразования.

2.3. UART

В данном проекте необходимо организовать обмен данными между контроллером «К1986*BE92QI*» и «*ESP-12F*» (см. приложение 3). Для этого был реализован символьный обмен*:*

Таблица 2.3.1. Символьный обмен контроллеров

Запрос (*ESP-12F*)	**Ответ *(*К1986*BE92QI)***
I(n) ((n) – номер стенда*)*	Значение тока на заданном стенде
S(n)_(s) *((n)* – номер стенда, *(s)*-состояние вкл./выкл.*)*	Включить/выключить стенд
All.I	Получить значение тока со всех стендов

2.3.1. Инициализация *UART* «К1986*BE92QI*»

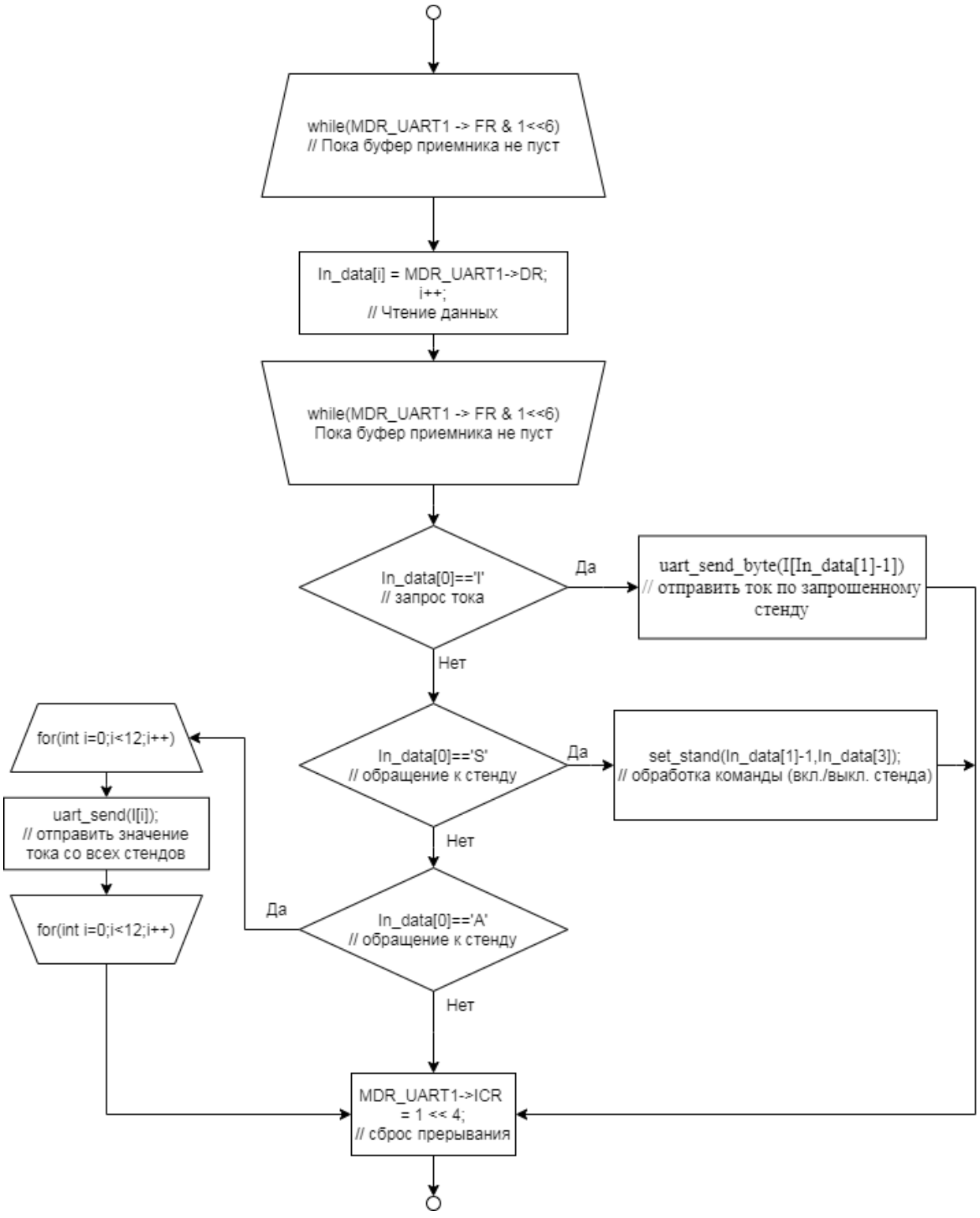
Для инициализации приемопередатчика необходимо порт «*PB5*» и «*PB6*» *(UART1)* настроить следующим образом*:*

Таблица 2.3.2. Инициализация модуля *UART1*

Регистр		**Состояние**
FUNC		Альтернативная функция
ANALOG		Цифровой порт
PWR		Максимально быстрый фронт
UART_CLOCK (регистр управление тактовой частотой)		Вкл. тактирование
		Не устанавливать делитель UART1
		Не устанавливать делитель *UART2*
		Разрешить тактирование *UART1*
		Запретить тактирование *UART2*
Параметры делителя при частоте 8 МГц	*IBRD*	= 4 Целая часть делителя скорости
	FBRD	=22 Дробная часть делителя скорости
		Работа без проверки четности
		Бит четности отключен
		Количество стоповых бит = 1

И скорости 115200 бод	*LCR_H*	Буфер *FIFO* выключен
		Размер кадра 8 бит
		Передача бита четности запрещена
CR		Разрешение приемопередатчика
IMSC		Разрешение прерывания от приемника

Прерывание *UART* возникает при поступлении байта данных в буфер приемника *UART*1. Программа выполняет алгоритм (см. блок-схему 2.3.1) цель которого обработать полученный запрос и сбросить прерывание.



Блок-схема 2.3.1. Обработка прерывания

3. Программное обеспечение для микроконтроллера «*ESP-12F*»

3.1. Настройка *UART*

На основе схемы печатной платы «МилКиТЭС» (см. рис. 3.1.1) были проинициализирован 21 и 22 дискретный канал*:*

```
SoftwareSerial softSerial(21,22);
```

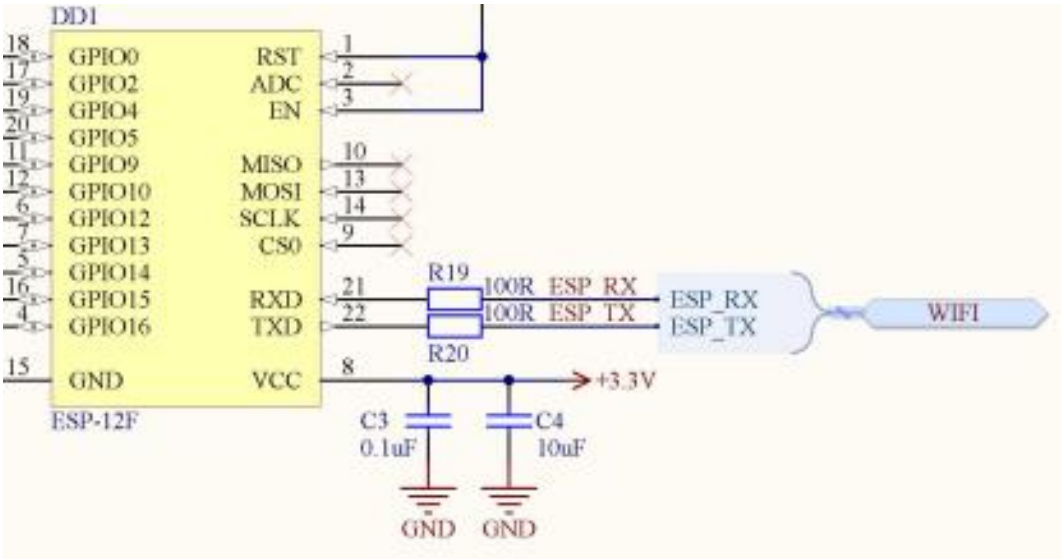
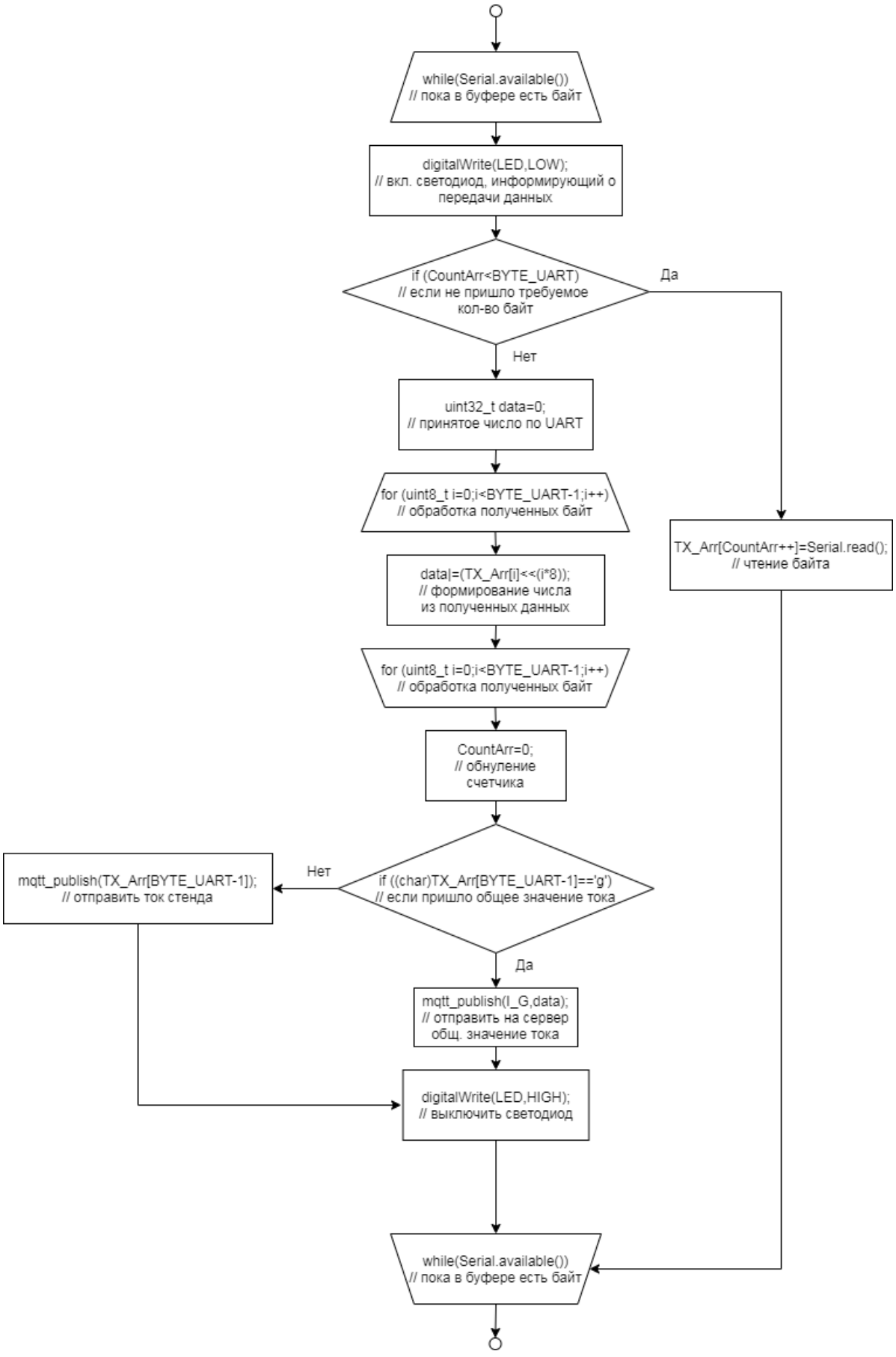


Рис. 3.1.1. Подключение *UART*

Для чтения байта по *UART* в бесконечном цикле программы (см. приложение 8) отслеживается наличие байта в буфере приемника, в случае поступления байта вызывается функция «*data_UART*» (см. блок-схему 3.1.1).



Блок-схема 3.1.1. Принятие данных по *UART*

3.2. Подключение к *MQTT* брокеру

Для подключения к серверу необходимо указать следующие данные [8]:

- имя сервера;
- порт для подключения к серверу;
- логин от сервера;
- пароль от сервера.

В бесконечном цикле программы отслеживается условия подключения к *MQTT*-серверу, если оно отсутствует, то программа снова подключается к серверу и подписывается на топики, в которые в дальнейшем будут поступать данные.

Таким образом, имея все необходимые функции для работы с *MQTT* сервером, остается настроить клиент. Существует большое множество таких клиентов, например, приложение на телефон «*IoT MQTT Dashboard*», данное приложение позволит пользователю отправлять и получать данные на *ESP*.

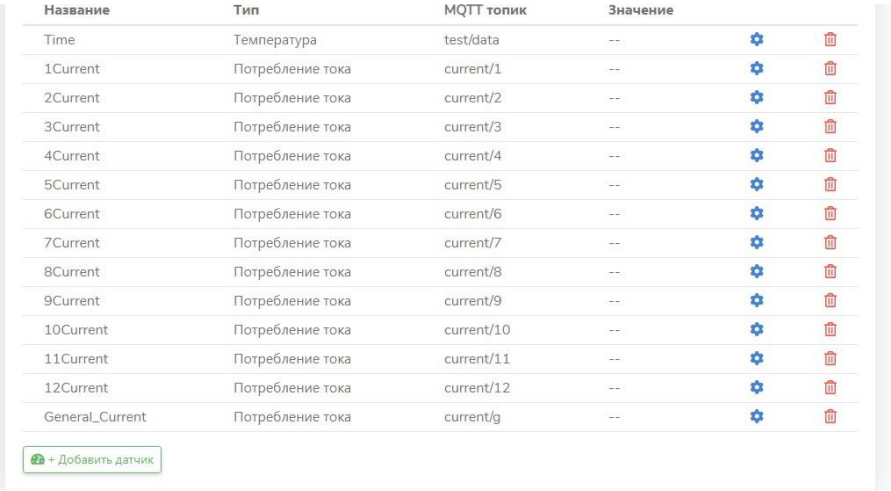


Рис. 3.2.1. Топики для публикации

Устройства

Название	Тип	MQTT топик	Статус		
LED	Лампа	test/led	--		
Stand_1	Выключатель	stand/1	--		
Stand_2	Выключатель	stand/2	--		
Stand_3	Выключатель	stand/3	--		
Stand_4	Выключатель	stand/4	--		
Stand_5	Выключатель	stand/5	--		
Stand_6	Выключатель	stand/6	--		
Stand_7	Выключатель	stand/7	--		
Stand_8	Выключатель	stand/8	--		
Stand_9	Выключатель	stand/9	--		
Stand_10	Выключатель	stand/10	--		
Stand_11	Выключатель	stand/11	--		
Stand_12	Выключатель	stand/12	--		

Рис. 3.2.2. Топики для отправки команды на главную плату

Блок-схема. 3.2.1. Алгоритм работы микроконтроллера «ESP-12F»

4. Контроль стендов

Измерение тока происходит с помощью микросхемы TPS561208 [4]. Она представляет собой дифференциальный усилитель сигнала (см. рис. 4.1). На измерительных резисторах (см. рис. 4.2) происходит падение напряжения. Измеряя падение на выходе микросхемы с помощью формулы (1) мы можем рассчитать ток, который проходит через измерительные резисторы.

$$I_{load} = \frac{V_{out}}{Rs*K}, \quad (1)$$

где I_{load} – ток нагрузки, V_{out} – выходное напряжение, Rs – сопротивление шунта, равное 0.1 Ом, K – коэффициент усиление сигнала (для измерения тока платформы МилКиТЭС K = 200).

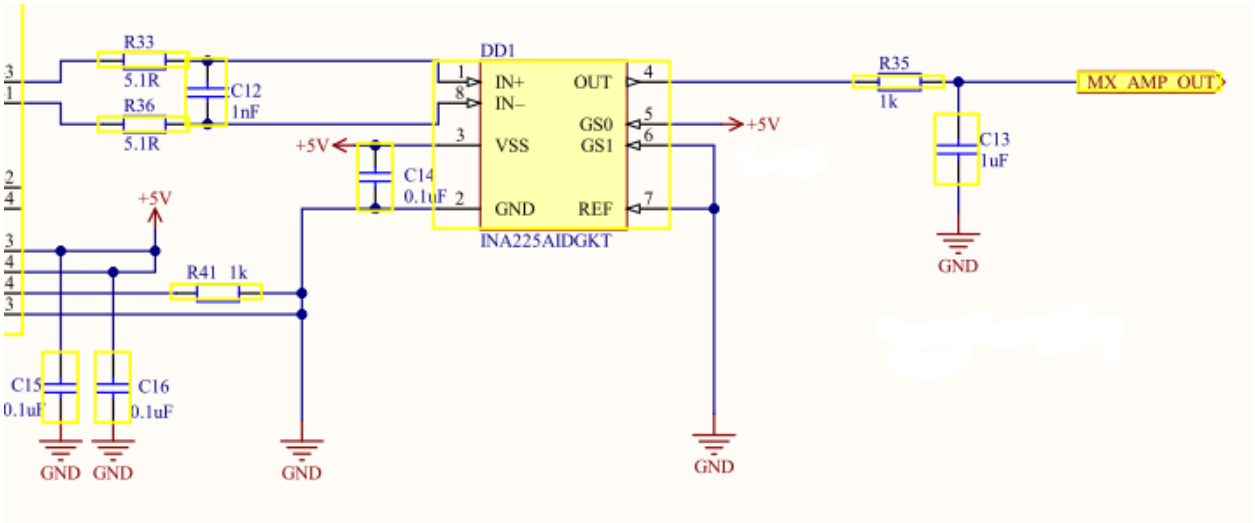


Рис. 4.1. Электрическая схема измерения тока платформы Страж

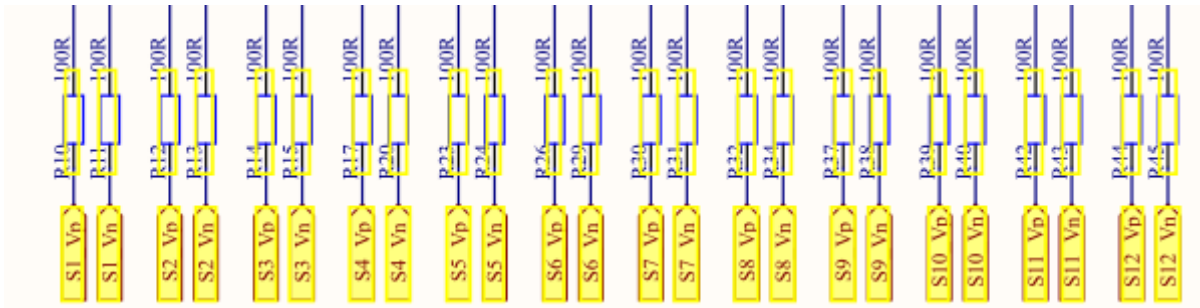


Рис. 4.2. Схема подключения измерительных резисторов

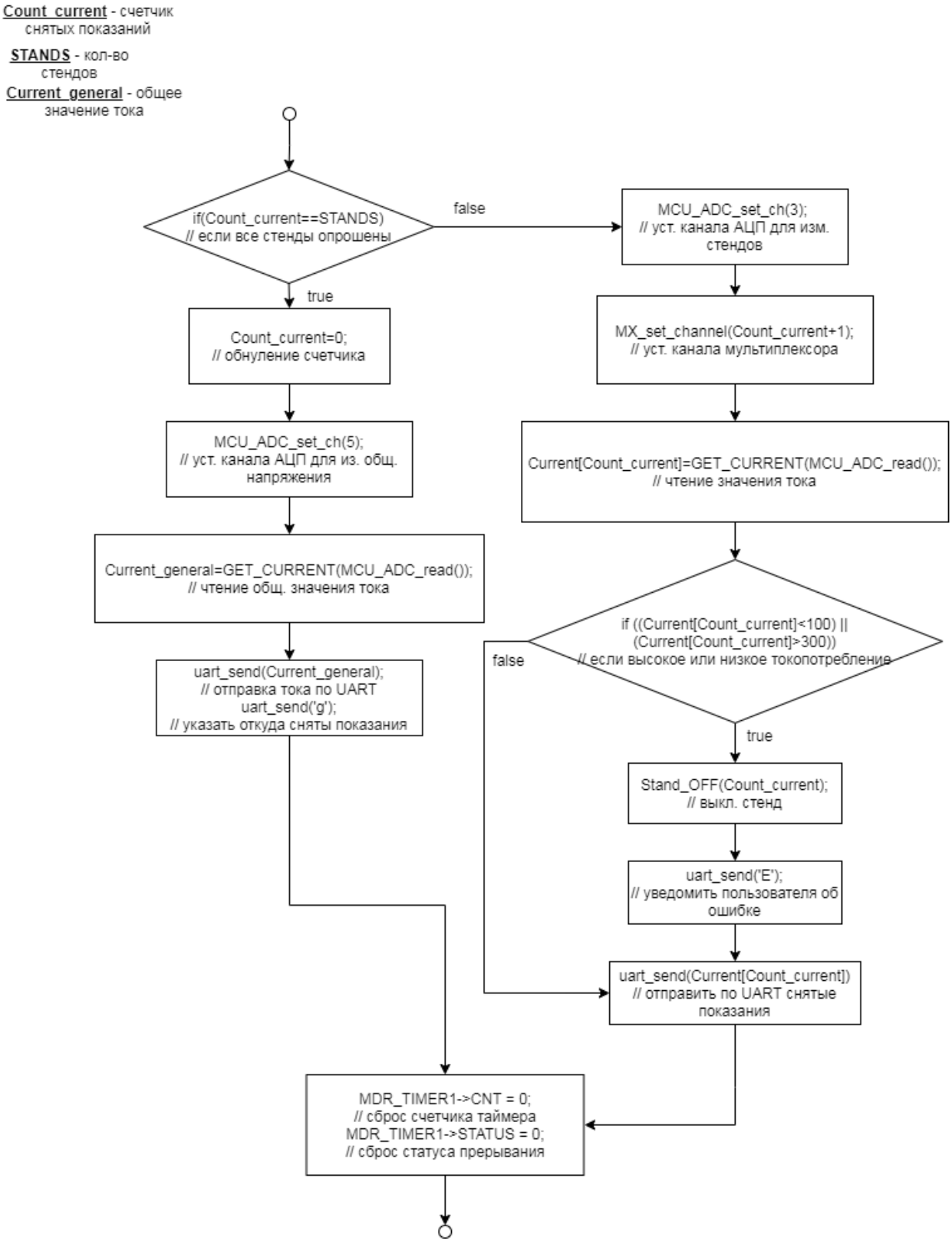
Для упрощения работы при написании ПО, на основе формулы (1), была написана вспомогательная директива:

$$\#define\ GET\_CURRENT(x)\ ((int32\_t)\ ((((float)x * 0.0008)\ /\ (0.1 * 200))\ *\ 1000))$$

В данной работе был задействован таймер, с целью задать период опроса стендов. В инициализации таймера [3 стр. 277] задействованы тактовые сигналы, параметры которых были заданы в блоке «Сигналы тактовой частоты»:

Таблица 4.1. Инициализация таймера

Регистр	**Значение**	**Описание**
MDR_RST_CLK->TIM_CLOCK	1<< 24	Включение тактирования Таймера 1
CNTRL	0x00000000	Режим счет – вверх
PSG	7999	Предделитель частоты
ARR	*TIM1_INTERRUPT*-1	Основание счета
CNT	0	Начальное значение счетчика
IE	2	Разрешить генерировать прерывание при *CNT=ARR*



Блок-схема. 4.1. Алгоритм обработки прерывания по Таймеру

5. Испытание проделанной работы

В данной работе для дистанционного контроля и мониторинга стендов использовалось *Android* приложение «*MQTT-Dashboard*». Приложение было настроено на отправление и получение данных от сервера.

5.1. Отправка снятых показаний на сервер

Цель: убедиться в корректной отправке токопотребления на сервер.

Вместо стенда был подключен резистор, с которого непрерывно снимаются показания и отправляются на сервер:

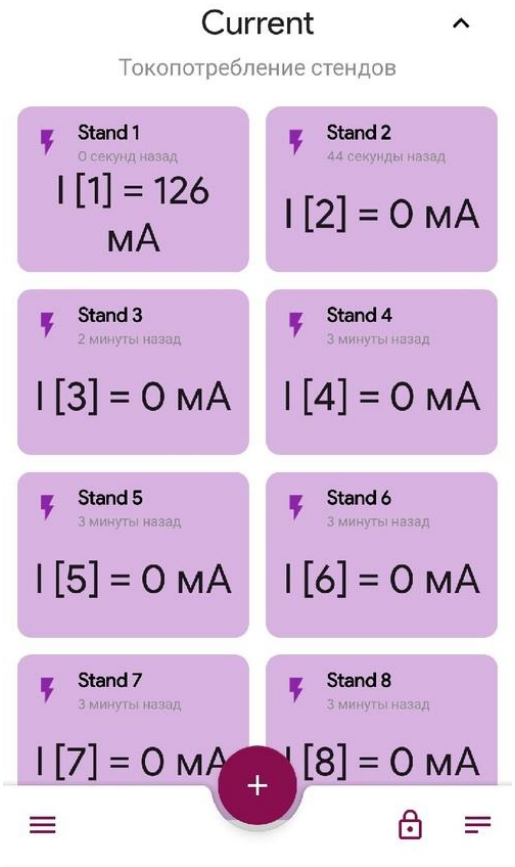


Рис. 5.1.1. Снятые показания с резистора

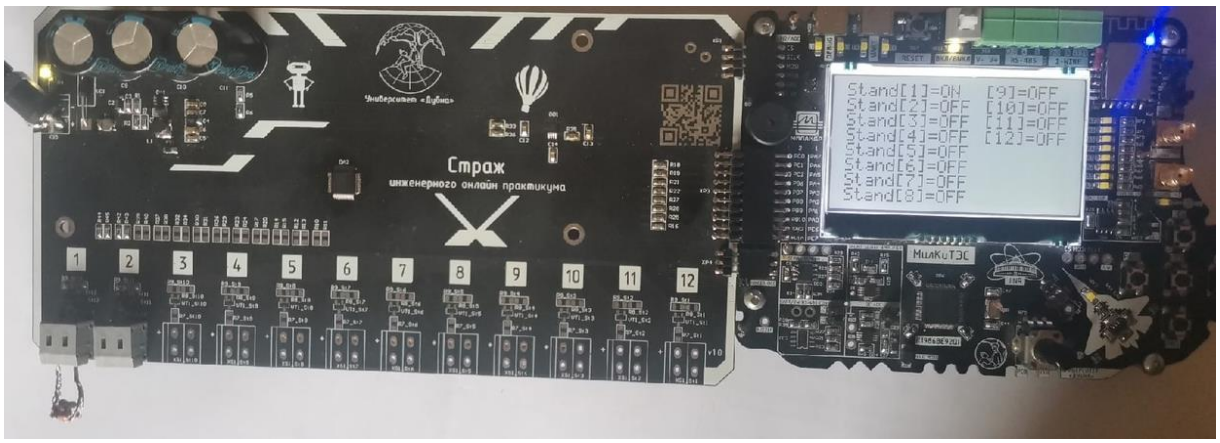


Рис. 5.1.2. Подключение резистора

Таким образом можно убедиться в том, что микроконтроллер успешно снимает показания с АЦП и отправляет их на сервер. В эксплуатации это выглядит следующим образом:

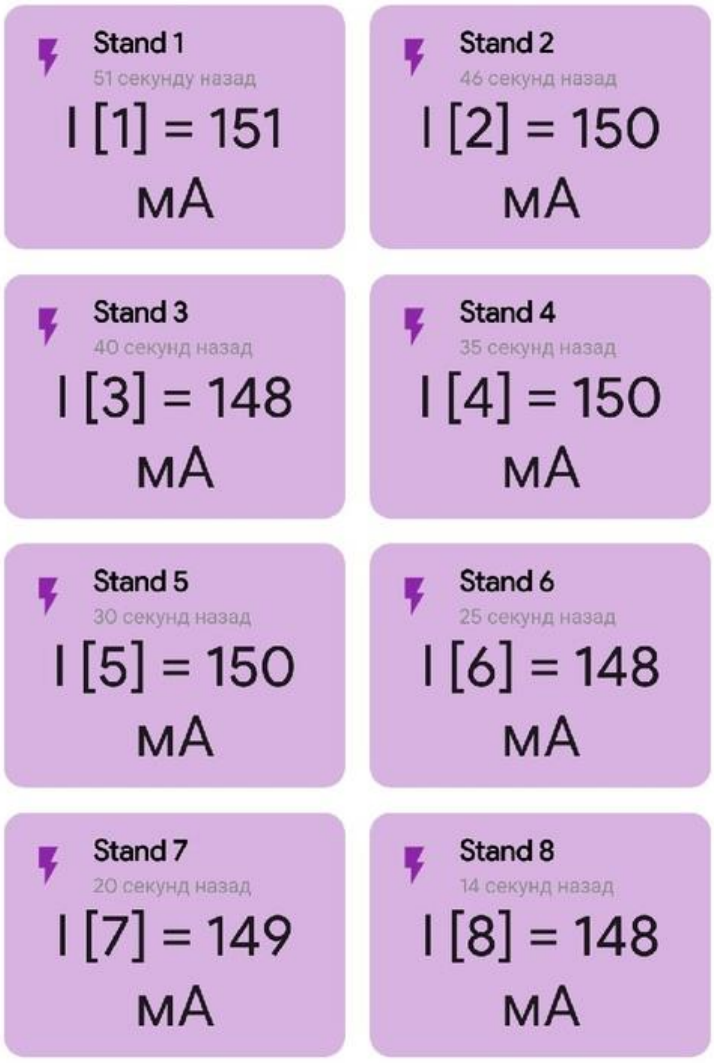


Рис. 5.1.3. Снятые показания со всех стендов

5.2. Дистанционный контроль стендов

На сервере имеются топики, управляющие питанием стендов. Всего 12 таких топиков, т.к. к комплексу «Страж» можно подключать только до 12 плат.

Цель: убедиться в корректной отправке и обработке запроса от сервера на плату, отвечающую за автоматизацию Стража.

Изначально все стенды включены и их состояние непрерывно отслеживает микроконтроллер:

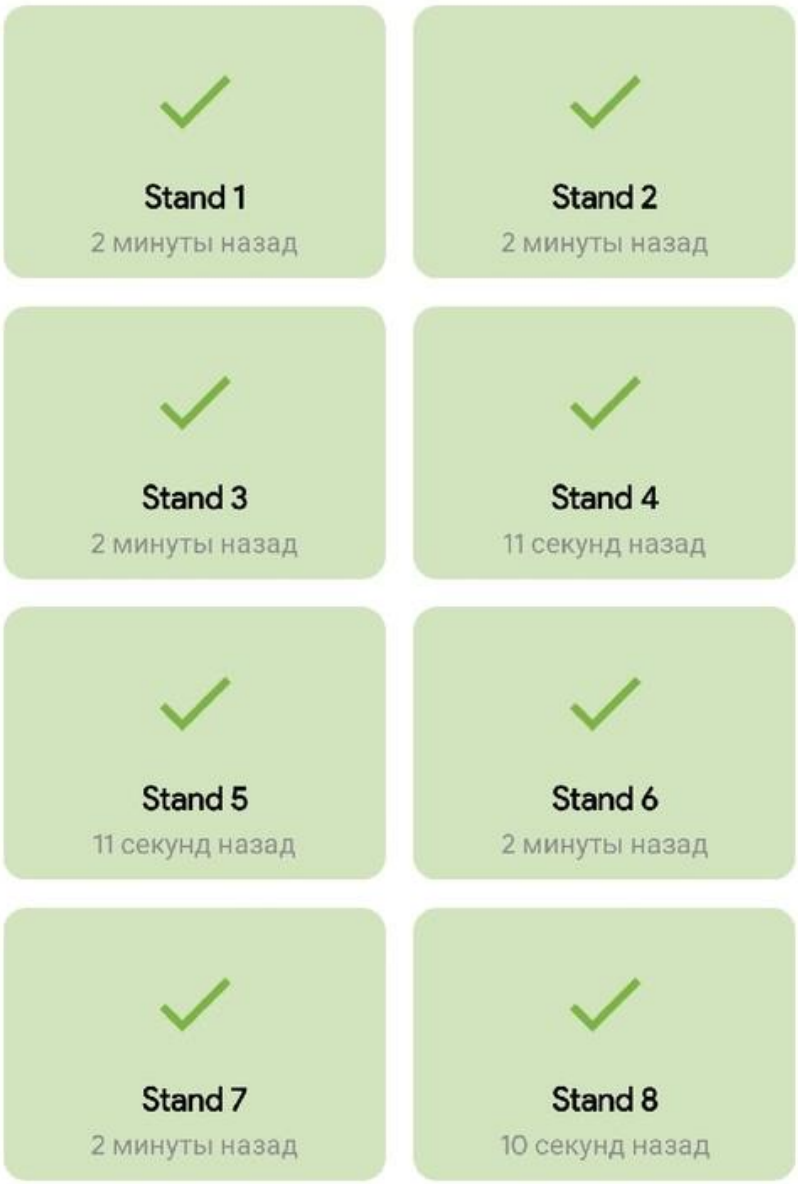


Рис. 5.2.1. Состояние стендов на сервере

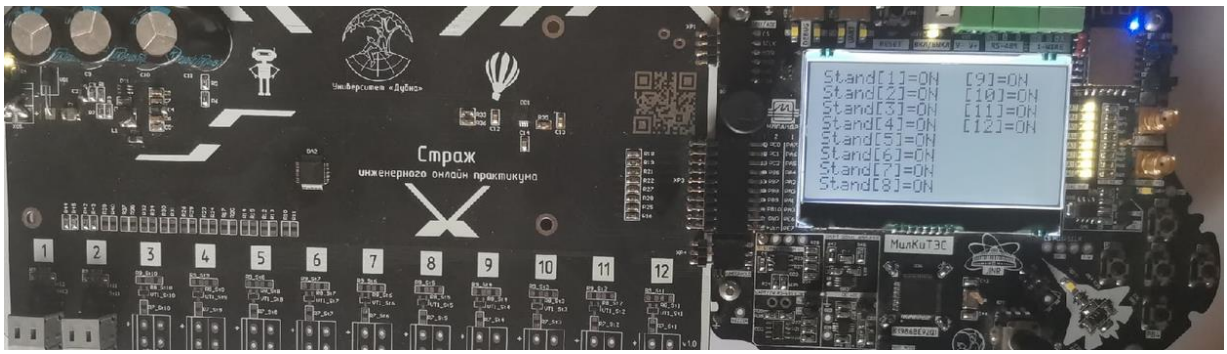


Рис. 5.2.2. Индикация дисплея

Далее через приложение был отключен 4,5 и 8 стенд:

Рис. 5.2.3. Измененное состояние стендов на сервере

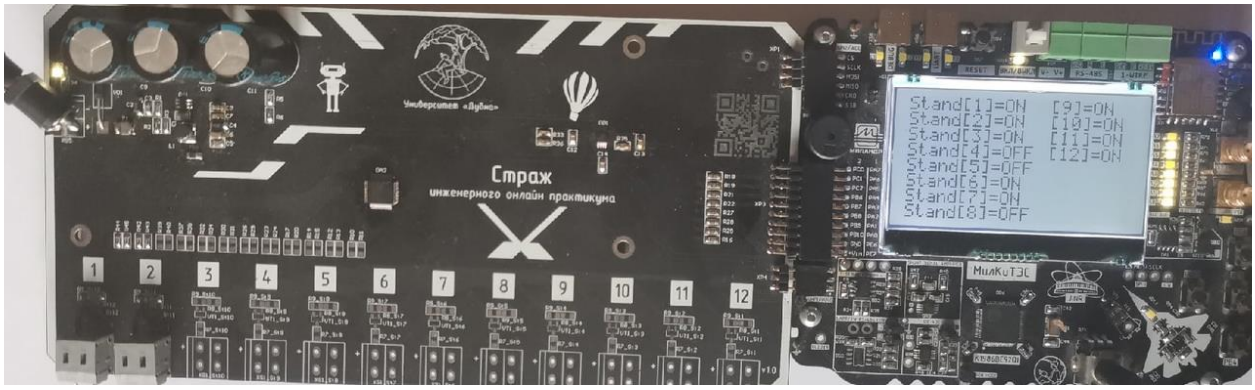


Рис. 5.2.4. Измененная индикация

После отправки запроса на микроконтроллер, состояние транзисторов, отвечающих за питание 4,5 и 8 стенда, моментально инвертировали свое состояние.

5.3. Вывод

Программное обеспечение моментально реагирует на запрос от сервера и непрерывно отправляет снятые показания.

Результат работы программы полностью удовлетворяет поставленную задачу.

Заключение

В результате выполнения бакалаврской работы разработан автоматизированный контроль печатных плат для учебно-демонстрационного комплекса «Страж». Система контроля была оснащена технологией «Интернета вещей», благодаря которой у пользователя появилась возможность дистанционного взаимодействия с учебным стендом.

В ходе работы были решены следующие задачи:

1. Автоматизированный контроль печатных плат:
    - инициализация АЦП и настройка обратной связи по напряжению;
    - установка канала мультиплексора;
    - контроль транзисторов, отвечающих за питание стендов;
    - запуск таймера и создание системы опроса печатных плат по прерыванию;
    - вывод снятых показаний на дисплей;
    - обработка показаний с АЦП.
2. Символьная система команд по UART:
    - настройка интерфейса UART на двух микроконтроллерах;
    - модель взаимодействия «Ведущий – ведомый»;
    - прерывание по поступлению байта в буфер приемника;
    - обработка алгоритма соответствующей команде.
3. Дистанционный контроль стенда:
    - настройка MQTT брокера;
    - передача показаний с АЦП на сервер;
    - прием команды от сервера (по типу – выключить стенд 1).

В процессе работы были приобретены и закреплены следующие навыки:

1. Изучены принципы работы следующих систем:
    - интерфейс UART – обмен данными двух микроконтроллеров;
    - интерфейс SPI – вывод информации на дисплей
    - оцифровка сигнала с помощью АЦП;
    - таймеры общего назначения.
2. Получены знания:
    - в программировании микроконтроллеров K1986BE92QI;

- в программировании микроконтроллеров: ESP12-F, ESP8266;
- в протоколах обмена данными технологией «Интернета вещей»;
- в организации удаленного доступа к электронному устройству.

3. Закреплены навыки работы:
    - с программатором «ST-Link»;
    - с программным кодом в среде Keil и Arduino;
    - с документациями и даташитами;
    - в отладке микроконтроллера;
    - с контрольно-измерительным оборудованием (мультиплексором, осцилогрофом);

Список литературы

1. Сахаров Ю.С. Горбунов Н.В. Люосев Д.А. Понкин Д.О. Шириков И.В. Основы программирования микроконтроллеров серии 1986BE9X в среде Keil uVision: учебное пособие. Университет Дубна, 2017. – 134с;
2. Проток MQTT [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/463669/;
3. Спецификация микроконтроллеров серии 1986BE9x, К1986BE9x, К1986BE92QI, К1986BE92QC, К1986BE91H4. ЗАО «ПКК Миландр» − Версия 3.4.3 от 22.07.2013: http://milandr.ru/uploads/Products/product_80/spec_seriya_1986BE9x.pdf;
4. *Datasheet* на TPS561208 [Электронный ресурс] – Режим доступа: https://www.ti.com/lit/ds/symlink/tps561201.pdf?ts=1623403111660&ref_url=https%253A%252F%252Fwww.google.com%252F;
5. Протоколы передачи данных *IoT* [Электронный ресурс] – Режим доступа: https://iot.ru/wiki/protokoly-peredachi-dannykh-iot;
6. Спецификация на ADG726 (мультиплексор) [Электронный ресурс] – Режим доступа: https://www.analog.com/media/en/technical-documentation/data-sheets/ADG726_732.pdf;
7. Репозиторий по программированию микроконтроллеров серии 1986BE9x, К1986BE9x, К1986BE92QI, К1986BE92QC, К1986BE91H4 [Электронный ресурс] – Режим доступа: https://github.com/owlatarms/mpb;
8. ESP8266 управление через интернет по протоколу MQTT [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/393277/;
9. *Datasheet* на *RTR*025*N*03*TL* [Электронный ресурс] – Режим доступа: https://pdf1.alldatasheet.com/datasheet-pdf/view/521031/ROHM/RTR025N03TL.html;
10. *Datasheet* на *INA225AIDGKT* [Электронный ресурс] – Режим доступа: https://pdf1.alldatasheet.com/datasheet-pdf/view/737768/TI/INA225AIDGKT.html.

Keil: main.c

```c
#include "main.h"   // подключение заголовка микроконтроллера
//#define ESP
#define Milandr

uint8_t stand[STANDS];
uint8_t In_data;  // входные данные

#define DELAY_ADC 50  // задержка между измерением показаний с АЦП

// Основная функция
int main(void)
{
  #ifdef Milandr
  Init_per();
  while (true)
  {
    Indication();
  }
  #endif

  #ifdef ESP
  while(true){}
  #endif
}
```

```c
/**
  * @brief      Прерывание UART1
  * @detailed   Прерывание возникает при поступлении 1 байта
  * @param      Функция не принимает параметры
  * @return     Функция не возвращает параметры
  *  */
void UART1_IRQHandler(void)
{
  while(MDR_UART1->FR & 1<<6)
  {
    In_data = MDR_UART1->DR;          // чтение данных

    // Обработка команды.             По UART были замечены помехи с этой целью были добавлены условия
    if (((In_data/10)<=STANDS)&&((In_data/10)>0))
    {
      if (In_data%10==1)
      {
        inquiry_stand(In_data);
        stand[(In_data/10)-1]=1;
      }
      else if (In_data%10==0)
      {
        inquiry_stand(In_data);
        stand[(In_data/10)-1]=0;
      }
    }

    //stand[(In_data/10)-1]=In_data%10; // обработка команды
  }
  MDR_UART1->ICR = 1 << 4;            // сброс прерывания
}
```

Keil: main.c

```
void Indication() // ф-я вывода состояния стэндов
{
  uint8_t page=0;
  for (uint8_t i=0;i<STANDS;i++)
  {
    if (page<8)
    {
      LCD_page_set(page++);
      LCD_column_set(0);
      LCD_print_text("Stand[");
      LCD_print_num(i+1);
      LCD_print_text("]=");
      if (stand[i]==1)
        LCD_print_text("ON ");
      else
        LCD_print_text("OFF");
    }
    else
    {
      LCD_page_set(page++-8);
      LCD_column_set(80);
      LCD_print_text("[");
      LCD_print_num(i+1);
      LCD_print_text("]=");
      if (stand[i]==1)
        LCD_print_text("ON ");
      else
        LCD_print_text("OFF");
    }
  }
}
```

```
/**
 * @brief      Обработка полученного запроса по UART
 * @detailed   Ф-я выполняет полученную команду по UART
 * @param      @data - полученные запрос
 * @return     Функция не возвращает параметры
 **/
void inquiry_stand(uint8_t data)
{
  if (((In_data/10)<=STANDS)&&((In_data/10)>0)) // если требуется вкл./выкл. стенд
  {
    switch(In_data/10)
    {
      case 1:                       // Стенд 1
        if (In_data%10)
          S1_EN;
        else
          S1_DIS;
        break;

      case 2:                       // Стенд 2
        if (In_data%10)
          S2_EN;
        else
          S2_DIS;
        break;

      case 3:                       // Стенд 3
        if (In_data%10)
          S3_EN;
        else
          S3_DIS;
        break;

      case 4:                       // Стенд 4
        if (In_data%10)
          S4_EN;
```

Keil: main.c

```
123       case 4:                          // Стенд 4
124         if (In_data%10)
125           S4_EN;
126         else
127           S4_DIS;
128         break;
129
130       case 5:                          // Стенд 5
131         if (In_data%10)
132           S5_EN;
133         else
134           S5_DIS;
135         break;
136
137       case 6:                          // Стенд 6
138         if (In_data%10)
139           S6_EN;
140         else
141           S6_DIS;
142         break;
143
144       case 7:                          // Стенд 7
145         if (In_data%10)
146           S7_EN;
147         else
148           S7_DIS;
149         break;
150
151       case 8:                          // Стенд 8
152         if (In_data%10)
153           S8_EN;
154         else
155           S8_DIS;
156         break;
```

```
157
158       case 9:                          // Стенд 9
159         if (In_data%10)
160           S9_EN;
161         else
162           S9_DIS;
163         break;
164
165       case 10:                         // Стенд 10
166         if (In_data%10)
167           S10_EN;
168         else
169           S10_DIS;
170         break;
171
172       case 11:                         // Стенд 11
173         if (In_data%10)
174           S11_EN;
175         else
176           S11_DIS;
177         break;
178
179       case 12:                         // Стенд 12
180         if (In_data%10)
181           S12_EN;
182         else
183           S12_DIS;
184         break;
185     }
186   }
187 }
```

Keil: main.h

main.c* | main.h | UART.h | Straj.h | Init.h | ADC.h | Timer.h | 1986VE9x.h

```
#include <1986VE9x.h>                    // Подключение заголовка микроконтроллера
#define   F_CPU 8000000                  // Указание тактовой частоты МК
#define STANDS  12                       // кол-во стэндов
#include  "milkites_delay.h"             // Подключение библиотеки задержек
#include  "milkites_display.h"           // Подключение библиотеки дисплея
#include  "UART.h"                       // UART
#include  "ADC.h"                        // Поключение функционала АЦП
#include  "Straj.h"                      // Подключения функционала мультиплексора
#include  "Timer.h"                      // Подлючения функционала таймеров
#include  "Init.h"                       // Инициализация МК

#define LCD_led_en  MDR_PORTE->RXTX |=  (1<<2)  // вкл. подсветки
#define LCD_led_dis  MDR_PORTE->RXTX &= ~(1<<2) // выкл. подсветки

void Indication(); // ф-я вывода состояния стэндов
void inquiry_stand(uint8_t data);// обработка полученного запроса по UART
```

Keil: UART.h

```c
/**
  * @brief      Отправка байта данных по UART1
  * @detailed
  * @param      @byte - байт данных для отправки по UART1
  * @return     Функция не возвращает параметры
  */
void uart_send_byte(uint32_t byte)
{
    // ожидание готовности UART1 для передачи байта данных
    while (MDR_UART1->FR & (1 << 5)) { }
    MDR_UART1->DR = byte; // отправка байта данных
}
```

```c
/**
  * @brief      Инициализация модуля UART1
  * @detailed
  * @param      Функция не принимает параметров
  * @return     Функция не возвращает параметры
  */
void uart_init(void)
{
  // режим работы порта PB5, PB6 - UART1 (Режим работы вывода порта - альтернативная ф-я)
  MDR_PORTB->FUNC  |= ((2 << 12) | (2 << 10));

  MDR_PORTB->ANALOG |= ((1 << 6) | (1 << 5));  // цифровой порт
  MDR_PORTB->PWR   |= ((3 << 12) | (3 << 10)); // максимально быстрый фронт

  MDR_RST_CLK->PER_CLOCK |= (1 << 6);          // вкл. тактирование UART1
  MDR_RST_CLK->UART_CLOCK = (0                 // установка делителя UART1 = 1
                           | (0 << 8)          // установка делителя UART2 = 1
                           | (1 << 24)         // разрешение тактирования UART1
                           | (0 << 25));       // запрет тактовой частоты UART2

  // Параметры делителя при частоте = 8 МГц и скорости 115200
  MDR_UART1->IBRD = 4;                         // целая часть делителя скорости
  MDR_UART1->FBRD = 22;                        // дробная часть делителя скорости
  MDR_UART1->LCR_H = ( (0 << 1)                // работа без проверки четности
                     | (0 << 2)                // бит четности откличен
                     | (0 << 3)                // кол-во стоповых бит = 1
                     | (0 << 4)                // буфер FIFO выключен
                     | (3 << 5)                // размер кадра - 8 бит
                     | (0 << 7));              // передача бита четности запрещена

  // передатчик и приемник разрешен, разрешение приемопередатчика UART1
  MDR_UART1->CR = ( (1 << 8) | (1 << 9) | 1);

  MDR_UART1->IMSC = 1 << 4;  // разрешение прерывания от приемника UARTR1XINTR
  NVIC_EnableIRQ(UART1_IRQn); // разрешение прерывания от модуля UART1
}
```

Keil: Straj.h

```
//---------------вкл./выкл. необходимого пина----------------------
#define PINA_ON(x)   MDR_PORTA->RXTX |=  (1<<x)  // вкл. пин x  PORTA
#define PINA_OFF(x)  MDR_PORTA->RXTX &= ~(1<<x) // выкл.пин x  PORTA

#define PINB_ON(x)   MDR_PORTB->RXTX |=  (1<<x)  // вкл. пин x  PORTB
#define PINB_OFF(x)  MDR_PORTB->RXTX &= ~(1<<x) // выкл.пин x  PORTB

#define PINC_ON(x)   MDR_PORTC->RXTX |=  (1<<x)  // вкл. пин x  PORTC
#define PINC_OFF(x)  MDR_PORTC->RXTX &= ~(1<<x) // выкл.пин x  PORTC

#define PIND_ON(x)   MDR_PORTA->RXTX |=  (1<<x)  // вкл. пин x  PORTD
#define PIND_OFF(x)  MDR_PORTA->RXTX &= ~(1<<x) // выкл.пин x  PORTD

#define PINF_ON(x)   MDR_PORTF->RXTX |=  (1<<x)  // вкл. пин x  PORTF
#define PINF_OFF(x)  MDR_PORTF->RXTX &= ~(1<<x) // выкл.пин x  PORTF

#define PINE_ON(x)   MDR_PORTE->RXTX |=  (1<<x)  // вкл. пин x  PORTE
#define PINE_OFF(x)  MDR_PORTE->RXTX &= ~(1<<x) // выкл.пин x  PORTE
//-----------------------------------------------------------------

/*----------------Директивы "Страж"-----------------------------------------
#define MX_EN     PIND_OFF(2)            // вкл. мультиплексор
#define MX_DIS    PIND_ON(2)             // выкл. мультиплексор

#define MX_CSa_EN   PINF_ON(7)           // 1 в MX_CSa (PE7)
#define MX_CSa_DIS  PINF_OFF(7)          // 0 в MX_CSa (PE7)

#define MX_CSb_EN   PINF_ON(0)           // 1 в MX_CSb (PE0)
#define MX_CSb_DIS  PINF_OFF(0)          // 0 в MX_CSb (PE0)
```

```
#define S12_EN     PINA_ON(0)            // ВКЛ.   стенд 1
#define S12_DIS    PINA_OFF(0)           // ВЫКЛ.  стенд 1
#define S11_EN     PINA_ON(1)            // ВКЛ.   стенд 2
#define S11_DIS    PINA_OFF(1)           // ВЫКЛ.  стенд 2
#define S10_EN     PINA_ON(2)            // ВКЛ.   стенд 3
#define S10_DIS    PINA_OFF(2)           // ВЫКЛ.  стенд 3
#define S9_EN     PINA_ON(3)             // ВКЛ.   стенд 4
#define S9_DIS    PINA_OFF(3)            // ВЫКЛ.  стенд 4
#define S8_EN     PINA_ON(4)             // ВКЛ.   стенд 5
#define S8_DIS    PINA_OFF(4)            // ВЫКЛ.  стенд 5
#define S7_EN     PINA_ON(5)             // ВКЛ.   стенд 6
#define S7_DIS    PINA_OFF(5)            // ВЫКЛ.  стенд 6
#define S6_EN     PINA_ON(6)             // ВКЛ.   стенд 7
#define S6_DIS    PINA_OFF(6)            // ВЫКЛ.  стенд 7
#define S5_EN     PINA_ON(7)             // ВКЛ.   стенд 8
#define S5_DIS    PINA_OFF(7)            // ВЫКЛ.  стенд 8
#define S4_EN     PINB_ON(8)             // ВКЛ.   стенд 9
#define S4_DIS    PINB_OFF(8)            // ВЫКЛ.  стенд 9
#define S3_EN    PINB_ON(9)              // ВКЛ.   стенд 10
#define S3_DIS   PINB_OFF(9)             // ВЫКЛ.  стенд 10
#define S2_EN    PINB_ON(10)             // ВКЛ.   стенд 11
#define S2_DIS   PINB_OFF(10)            // ВЫКЛ.  стенд 11
#define S1_EN    PINB_ON(7)              // ВКЛ.   стенд 12
#define S1_DIS   PINB_OFF(7)             // ВЫКЛ.  стенд 12
//-----------------------------------------------------------------
/*-----------------------------------------------------------------
```

Keil: Straj.h

```
/**
  * @brief      Установка канала мультиплексора
  * @detailed
  * @param      @channel - номер стенда
  * @return     Функция не возвращает параметры
  */
void MX_set_channel(uint8_t channel)
{
  switch(channel)
  {
    case 12:                //Стенд 12  0000
      PINC_OFF(0);
      PINC_OFF(1);
      PINC_OFF(2);
      PINB_OFF(6);
      break;
    case 11:                //Стенд 11  0001
      PINC_OFF(0);
      PINC_OFF(1);
      PINC_OFF(2);
      PINB_ON(6);
      break;
    case 10:                //Стенд 10  0010
      PINC_OFF(0);
      PINC_OFF(1);
      PINC_ON(2);
      PINB_OFF(6);
      break;
    case 9:                 //Стенд 9  0011
      PINC_OFF(0);
      PINC_OFF(1);
      PINC_ON(2);
      PINB_ON(6);
      break;
```

```
    case 8:                 //Стенд 8  0100
      PINC_OFF(0);
      PINC_ON(1);
      PINC_OFF(2);
      PINB_OFF(6);
      break;
    case 7:                 //Стенд 7  0101
      PINC_OFF(0);
      PINC_ON(1);
      PINC_OFF(2);
      PINB_ON(6);
      break;
    case 6:                 //Стенд 6  0110
      PINC_OFF(0);
      PINC_ON(1);
      PINC_ON(2);
      PINB_OFF(6);
      break;
    case 5:                 //Стенд 5  0111
      PINC_OFF(0);
      PINC_ON(1);
      PINC_ON(2);
      PINB_ON(6);
      break;
    case 4:                 //Стенд 4  1000
      PINC_ON(0);
      PINC_OFF(1);
      PINC_OFF(2);
      PINB_OFF(6);
      break;
    case 3:                 //Стенд 3 1001
      PINC_ON(0);
      PINC_OFF(1);
      PINC_OFF(2);
      PINB_ON(6);
      break;
    case 2:                 //Стенд 2 1010
```

Keil: Straj.h

```
153                 break;
154             case 2:                 // Стенд 2 1010
155                 PINC_ON(0);
156                 PINC_OFF(1);
157                 PINC_ON(2);
158                 PINB_OFF(6);
159                 break;
160             case 1:                 // Стенд 1 1011
161                 PINC_ON(0);
162                 PINC_OFF(1);
163                 PINC_ON(2);
164                 PINB_ON(6);
165                 break;
166         }
167     }
168
169
```

```
172 /**
173   * @brief    Инициализация "Страж"
174   * @detailed   Инициализация портов для взаимодействия
175   *             со "Стражем"
176   * @param    Функция не принимает параметров
177   * @return   Функция не возвращает параметры
178   **/
179 void Init_Straj(void)
180 {
181   /*
182   * PA0..PA7,PB8..PB10,PF5  - на выход, управление питанием с 1 по 12
183   * PC0-PC2,PB6             - на выход, установка канала мультиплексора
184   * PD2                     - на выход, вкл./выкл. мультиплексора
185   */
186
187   // подсветка дисплея управляется выводом PE2
188   MDR_PORTE->OE    = 0xff37; // биты 7,6,3 PORTE - входы, др. - выходы
189   MDR_PORTE->FUNC  = 0x0000; // функция - порт, основная функция
190   MDR_PORTE->PWR   = 0xff37; // макс. быстрый фронт
191   MDR_PORTE->ANALOG = 0xffff; // режим работы порта - цифровой ввод/вывод
192
193   /*-------------- Управление питанием ---------------------------------------*/
194   //-------------- PORTA ----------------------------------------
195   // Конфигурация линий PA0...PA7
196   MDR_PORTA->OE    |= 0x000000FF; // Вывод          (Направление передачи)
197   MDR_PORTA->FUNC  &= ~0x0000FFFF; // Ввод-вывод     (Функция)
198   MDR_PORTA->ANALOG |= 0x000000FF; // Цифровой       (Режим работы)
199   MDR_PORTA->PULL  &= ~0x00FF00FF; // Отключена      (Подтяжка)
200   MDR_PORTA->PD    &= 0x00FF00FF; // Драйвер        (Управление выводом)
201   MDR_PORTA->PWR   |= 0x0000FFFF; // Высокая        (Крутизна фронтов)
202   MDR_PORTA->GFEN  &= ~0x000000FF; // Не используется (Цифровой фильтр)
203   //------------------------------------------------------------------
204
```

```
205   //-------------- PORTB ----------------------------------------
206   // Конфигурация линий PB8..PB10
207   MDR_PORTB->OE    |= (1<<8)     // Вывод     PB8  (Направление передачи)
208                     |  (1<<9)     // Вывод     PB9
209                     |  (1<<10);   // Вывод     PB10
210   MDR_PORTB->FUNC  &= ~(3<<(2*8)); // Ввод-вывод PB8  (Функция)
211   MDR_PORTB->FUNC  &= ~(3<<(2*9)); // Ввод-вывод PB9
212   MDR_PORTB->FUNC  &= ~(3<<(2*10));// Ввод-вывод PB10
213   MDR_PORTB->ANALOG |= (1<<8)     // Цифровой   PB8  (Режим работы)
214                     |  (1<<9)     // Цифровой   PB9
215                     |  (1<<10);   // Цифровой   PB10
216   MDR_PORTB->PULL  &= ~(1<<8);    // Отключена  PB8  (Подтяжка) DOWN - подтяжка линии к потенциалу питания (3.3 В)
217   MDR_PORTB->PULL  &= ~(1<<24);   // Отключена  PB8  (Подтяжка) UP  - подтяжка линии к потенциалу земли   (0 В)
218   MDR_PORTB->PULL  &= ~(1<<9);    // Отключена  PB9  DOWN
219   MDR_PORTB->PULL  &= ~(1<<25);   // Отключена  PB9  UP
220   MDR_PORTB->PULL  &= ~(1<<10);   // Отключена  PB10 DOWN
221   MDR_PORTB->PULL  &= ~(1<<26);   // Отключена  PB10 UP
222   MDR_PORTB->PD    &= ~(1<<8);    // Драйвер    PB8  (Управление выводом)
223   MDR_PORTB->PD    &= ~(1<<24);   // Т.Г. Dis.  PB8  Триггер Шмитта(ТГ) - Отключен (Dis.)
224   MDR_PORTB->PD    &= ~(1<<9);    // Драйвер    PB9
225   MDR_PORTB->PD    &= ~(1<<25);   // Т.Г. Dis.  PB9
226   MDR_PORTB->PD    &= ~(1<<10);   // Драйвер    PB10
227   MDR_PORTB->PD    &= ~(1<<26);   // Т.Г. Dis.  PB10
228   MDR_PORTB->PWR   |= (3<<(2*8))  // Высокая    PB8  (Крутизна фронтов)
229                     |  (3<<(2*9))  // Высокая    PB9
230                     |  (3<<(2*10));// Высокая    PB10
231   MDR_PORTB->GFEN  &= ~(1<<8);    // Отключен   PB8  (Цифровой фильтр)
232   MDR_PORTB->GFEN  &= ~(1<<9);    // Отключен   PB9
233   MDR_PORTB->GFEN  &= ~(1<<10);   // Отключен   PB10
234   //------------------------------------------------------------------
235
```

```
237   //-------------- PORTB ----------------------------------------
238   MDR_PORTB->OE    |= (1<<6);     // Вывод     PB6  (Направление передачи)
239   MDR_PORTB->FUNC  &= ~(3<<(2*6)); // Ввод-вывод PB6  (Функция)
240   MDR_PORTB->ANALOG |= (1<<6);    // Цифровой   PB6  (Режим работы)
241   MDR_PORTB->PULL  &= ~(1<<6);    // Отключена  PB6  (Подтяжка) DOWN - подтяжка линии к потенциалу питания (3.3 В)
242   MDR_PORTB->PULL  &= ~(1<<22);   // Отключена  PB6  (Подтяжка) UP  - подтяжка линии к потенциалу земли   (0 В)
243   MDR_PORTB->PD    &= ~(1<<6);    // Драйвер    PB6  (Управление выводом)
244   MDR_PORTB->PD    &= ~(1<<22);   // Т.Г. Dis.  PB6  Триггер Шмитта(ТГ) - Отключен (Dis.)
245   MDR_PORTB->PWR   |= (3<<(2*6)); // Высокая    PB6  (Крутизна фронтов)
246   MDR_PORTB->GFEN  &= ~(1<<6);    // Отключен   PB6  (Цифровой фильтр)
247   //------------------------------------------------------------------
```

Keil: Straj.h

```
  //--------------PORTC-------------------------------------------
  // Конфигурация линий PC0..PC2
  MDR_PORTC->OE    |= (1<<0)      // Вывод     PC0  (Направление передачи)
                   |  (1<<1)      // Вывод     PC1
                   |  (1<<2);     // Вывод     PC2
  MDR_PORTC->FUNC  &= ~(3<<(2*0)); // Ввод-вывод PC0  (Функция)
  MDR_PORTC->FUNC  &= ~(3<<(2*1)); // Ввод-вывод PC1
  MDR_PORTC->FUNC  &= ~(3<<(2*2)); // Ввод-вывод PC2
  MDR_PORTC->ANALOG |= (1<<0)     // Цифровой  PC0  (Режим работы)
                    |  (1<<1)     // Цифровой  PC1
                    |  (1<<2);    // Цифровой  PC2
  MDR_PORTC->PULL  &= ~(1<<0);    // Отключена PC0  (Подтяжка) DOWN - подтяжка линии к потенциалу питания (3.3 В)
  MDR_PORTC->PULL  &= ~(1<<16);   // Отключена PC0  (Подтяжка) UP   - подтяжка линии к потенциалу земли   (0 В)
  MDR_PORTC->PULL  &= ~(1<<1);    // Отключена PC1  DOWN
  MDR_PORTC->PULL  &= ~(1<<17);   // Отключена PC1  UP
  MDR_PORTC->PULL  &= ~(1<<2);    // Отключена PC2  DOWN
  MDR_PORTC->PULL  &= ~(1<<18);   // Отключена PC2  UP
  MDR_PORTC->PD    &= ~(1<<0);    // Драйвер   PC0  (Управление выводом)
  MDR_PORTC->PD    &= ~(1<<16);   // Т.Г.- Dis. PC0  Триггер Шмитта(ТГ) - Отключен (Dis.)
  MDR_PORTC->PD    &= ~(1<<1);    // Драйвер   PC1
  MDR_PORTC->PD    &= ~(1<<17);   // Т.Г.- Dis. PC1
  MDR_PORTC->PD    &= ~(1<<2);    // Драйвер   PC2
  MDR_PORTC->PD    &= ~(1<<18);   // Т.Г.- Dis. PC2
  MDR_PORTC->PWR   |= (3<<(2*0))  // Высокая   PC0  (Крутизна фронтов)
                   |  (3<<(2*1))  // Высокая   PC1
                   |  (3<<(2*2)); // Высокая   PC2
  MDR_PORTC->GFEN  &= ~(1<<0);    // Отключен  PC0  (Цифровой фильтр)
  MDR_PORTC->GFEN  &= ~(1<<1);    // Отключен  PC1
  MDR_PORTC->GFEN  &= ~(1<<2);    // Отключен  PC2
  //-------------------------------------------------------------
```

```

  //--------------PORTF-------------------------------------------
  MDR_PORTF->OE    |= (1<<4);     // Вывод     PF4  (Направление передачи)
  MDR_PORTF->FUNC  &= ~(3<<(2*4)); // Ввод-вывод PF4  (Функция)
  MDR_PORTF->ANALOG |= (1<<4);    // Цифровой  PF4  (Режим работы)
  MDR_PORTF->PULL  &= ~(1<<4);    // Отключена PF4  (Подтяжка) DOWN - подтяжка линии к потенциалу питания (3.3 В)
  MDR_PORTF->PULL  &= ~(1<<20);   // Отключена PF4  (Подтяжка) UP   - подтяжка линии к потенциалу земли   (0 В)
  MDR_PORTF->PD    &= ~(1<<4);    // Драйвер   PF4  (Управление выводом)
  MDR_PORTF->PD    &= ~(1<<20);   // Т.Г.- Dis. PF4  Триггер Шмитта(ТГ) - Отключен (Dis.)
  MDR_PORTF->PWR   |= (3<<(2*4)); // Высокая   PF4  (Крутизна фронтов)
  MDR_PORTF->GFEN  &= ~(1<<4);    // Отключен  PF4  (Цифровой фильтр)
  //-------------------------------------------------------------
  /*-------------------------------------------------------------------------------*/

  /*--------------вкл./выкл. мультиплексора-------------------------------------*/
  // Конфигурация линии PD2
  MDR_PORTD->OE    |= (1<<2);     // Вывод     PD2  (Направление передачи)
  MDR_PORTD->FUNC  &= ~(3<<(2*2)); // Ввод-вывод PD2  (Функция)
  MDR_PORTD->ANALOG |= (1<<2);    // Цифровой  PD2  (Режим работы)
  MDR_PORTD->PULL  &= ~(1<<2);    // Отключена PD2  (Подтяжка) DOWN - подтяжка линии к потенциалу питания (3.3 В)
  MDR_PORTD->PULL  &= ~(1<<18);   // Отключена PD2  (Подтяжка) UP   - подтяжка линии к потенциалу земли   (0 В)
  MDR_PORTD->PD    &= ~(1<<2);    // Драйвер   PD2  (Управление выводом)
  MDR_PORTD->PD    &= ~(1<<18);   // Т.Г.- Dis. PD2  Триггер Шмитта(ТГ) - Отключен (Dis.)
  MDR_PORTD->PWR   |= (3<<(2*2)); // Высокая   PD2  (Крутизна фронтов)
  MDR_PORTD->GFEN  &= ~(1<<2);    // Отключен  PD2  (Цифровой фильтр)
  /*-------------------------------------------------------------------------------*/
}
```

Keil: Init.h

```
void Init_per()
{
  LED_Init();                    // Инициализация светодиодов

  MDR_RST_CLK->PER_CLOCK = 0xffffffff; // вкл. тактир. всей периферии МК

  MDR_PORTE->OE    = 0xff37; // биты 7,6,3 PORTE - входы, др. - выходы
  MDR_PORTE->FUNC  = 0x0000; // функция - порт, основная функция
  MDR_PORTE->PWR   = 0xff37; // макс. быстрый фронт
  MDR_PORTE->ANALOG = 0xffff; // режим работы порта - цифровой ввод/вывод

  delay_init();                  // инициализация системы задержек

  LCD_init();                    // инициализация дисплея
  LCD_clear();                   // очистка дисплея
  MDR_PORTE->RXTX |= (1<<2); // вкл. подсветки

  Init_Straj();                  // Инициализация "Страж"
  MCU_ADC1_init();               // Инициализация АЦП

  uart_init();                  // инициализация модуля UART
  MDR_UART1->IMSC = 1 << 4;   // инициализация прерывания от UART
  NVIC_EnableIRQ(UART1_IRQn);// разрешение прерывания от модуля UART1

  //-----------------Таймер1 измер. напряжения ------------------------------
  Timer1_init();                    // инициализация Таймера 1
  NVIC_EnableIRQ(Timer1_IRQn);      // разрешение прерывания от Таймера 1
  __enable_irq();                   // глобальное разрешение прерываний
  Timer1_start();                   // запуск Таймера 1
  NVIC_SetPriority (Timer1_IRQn, 3);  // Установка приоритета для таймера
  //-------------------------------------------------------------------------
}
```

Keil: ADC.h

```c
void MCU_ADC1_init()
{
  MDR_RST_CLK->PER_CLOCK |= RST_CLK_PCLK_ADC_Msk;          // вкл. тактирование АЦП
  // настройка конфигурации АЦП
  MDR_ADC->ADC1_CFG = ( 1 << ADC1_CFG_REG_ADON_Pos)        // Работа АЦП                    (включен)
                    |(0 << ADC1_CFG_REG_CLKS_Pos)          // Источник тактирования АЦП     (CPU)
                    |(0 << ADC1_CFG_REG_SAMPLE_Pos)        // Способ запуска АЦП            (однократный)
                    |(0 << ADC1_CFG_REG_CHS_Pos)           // Целевой канал преобразователя (уст. во время работы программы)
                    |(0 << ADC1_CFG_REG_CHCH_Pos)          // Режим последовательного переключения каналов (отключён)
                    |(0 << ADC1_CFG_REG_RNGC_Pos)          // Контроль границ преобразования (отключён)
                    |(0 << ADC1_CFG_M_REF_Pos)             // Источник опорного напряжения  (внутренний)
                    |(3 << ADC1_CFG_REG_DIVCLK_Pos)        // Делитель тактовой частоты АЦП (2^3 = 8)
                    |(0 << ADC1_CFG_SYNC_CONVER_Pos)       // Режим запуска двух АЦП (независимый)
                   //...Конфигурация датчика температуры и внутреннего источника напряжения 1.23 В
                    | (0 << ADC1_CFG_TS_EN_Pos)            // Работа блока датчика температуры и внутреннего источника напряжения 1.23 В (отключён)
                    | (0 << ADC1_CFG_TS_BUF_EN_Pos)        // Работа усилителя для датчика температуры и внутреннего источника напряжения 1.23 В (отключён)
                    | (0 << ADC1_CFG_SEL_TS_Pos)           // Преобразование сигнала с датчика температуры (отключено)
                    | (0 << ADC1_CFG_SEL_VREF_Pos)         // Преобразование сигнала с внутреннего источника напряжения 1.23 В (отключено)
                    | (0 << ADC1_CFG_TR_Pos)               // Подстройка напряжения внутреннего источника 1.23 В
                   //...Настройка задержек при преобразовании
                   //|(7 << ADC1_CFG_DELAY_GO_Pos)         // Дополнительная задержка при выборе канала (8 тактов ядра)
                    |(0 << ADC1_CFG_DELAY_ADC_Pos);        // Разность фаз между циклами преобразователей (не используется)

                   // ADC3 MX_OUT - напряжение на мультиплексоре
                   MDR_PORTD->OE    &=~(1 << 3); // настройка PD3 на вход
                   MDR_PORTD->ANALOG &=~(1 << 3); // перевод PD3 в аналоговый режим

                   // ADC4 ШУНТ - измерение напряжения главной платы
                   MDR_PORTD->OE    &=~(1 << 5); // настройка PD5 на вход
                   MDR_PORTD->ANALOG &=~(1 << 5); // перевод PD5 в аналоговый режим

                   // ADC5 12v/4 - общ. напряжение
                   MDR_PORTD->OE    &=~(1 << 5); // настройка PD5 на вход
                   MDR_PORTD->ANALOG &=~(1 << 5); // перевод PD5 в аналоговый режим
}
```

```c
/**
  * @brief     Установка канала АЦП
  * @detailed  3 канал - измерять напряжение на мультиплексоре
  *            4 канал - напряжение главной платы
  *            5 канал - измерять общ напряжение
  * @param     @channel - номер канала АЦП
  * @return    Функция не возвращает параметры
  */
void MCU_ADC_set_ch(uint8_t channel)
{
  // проверка номера канала и возврат в случае выхода из диапазона
  if (channel > 15) return;              // всего 16 каналов (с 0)
  MDR_ADC->ADC1_CFG  |= channel << 4;    // уст. канала АЦП
}

void MCU_ADC_start_conv(void)    // начало преобразования
{
  MDR_ADC->ADC1_CFG |= 1 << ADC1_CFG_REG_GO_Pos;  // Запись "1" начинает процесс преобразования
}
```

```c
/**
  * @brief     Чтение результатов преобразования АЦП
  * @detailed
  * @param     Не принимает параметров
  * @return    @ADC_data - результат преобразования
  */
uint32_t MCU_ADC_read(void)
{
  uint32_t ADC_data = 0;  // локальная для хранения результата преобр.

  MCU_ADC_start_conv();  // команда начала преобразования
  // пустой цикл - ожидание окончания преобразования
  while(!(MDR_ADC->ADC1_STATUS) & (1<<2)) { }
  ADC_data = MDR_ADC->ADC1_RESULT; // чтение результата преобразований
  // очистка битов содержащих номер канала преобразования - обнуление
  // старшего полубайта регистра
  ADC_data  =  ADC_data & 0x0FFF;
  return ADC_data; // возврат результата преобразования
}
```

Arduino

```
#define  F_CPU 8000000                  // Указание тактовой частоты МК
#include  "milkites_delay.h"            // Подключение библиотеки задержек
#include  "milkites_display.h"          // Подключение библиотеки дисплея
#define STANDS  12                      // кол-во стэндов
#define TIM1_INTERRUPT 5000             // Период опроса АЦП (мс.)
#define GET_CURRENT(x) ((int32_t) ((((float)x*0.0008) / (0.1*200)) * 1000))
#define DELAY_UART 5                    // задержка между отправкой байта в UART

volatile uint32_t Count_current=0;  // счетчик снятия показаний
volatile uint32_t Current[STANDS];  // токопотребление стендов
volatile uint32_t Current_general;  // значение общего тока

//------------------ Timer 1 -----------------------------------------------
void Timer1_init(void)
{ // настройка T1 на генерирование прерывания каждум секунду
  MDR_RST_CLK->TIM_CLOCK |= (1<< 24); // вкл. тактирование Таймера 1

  // режим счета - вверх,начальное значение - число из регистра CNT
  MDR_TIMER1->CNTRL = 0x00000000;
  MDR_TIMER1->PSG = 7999;  // предделитель частоты
  MDR_TIMER1->ARR = TIM1_INTERRUPT-1;  // основание счета = CNT + 1 = TIM1_INTERRUPT
  MDR_TIMER1->CNT = 0;     // начальное значение счетчика
  MDR_TIMER1->IE  = 2;     // разрешение генерир. прерывание при CNT=ARR
}
```

Arduino

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <SoftwareSerial.h>
#define STANDS 12                               // кол-во стендов

// UART
SoftwareSerial softSerial(21,22);               // объявление задействованных дискретных каналов      RX, TX
#define BYTE_UART 3                             // кол-во байт UART
uint32_t TX_Arr[BYTE_UART];                     // массив принятых байт
uint8_t CountArr=0;                             // кол-во приинятых байт
uint32_t Current_St[STANDS];                    // токопотребление на стендах

#define LED 2                                   // определяем пин светодиода
#define MQTT_client "ah76kkji"                  // произвольное название MQTT клиента, иногда требуется уникальное.

// настройки домашней сети
const char *ssid          = "xxxxx";            // название точки доступа
const char *pass          = "xxxx";    // пароль от точки доступа

// настройки для MQTT брокера
const char *mqtt_server   = "xxxx";     // адрес сервера MQTT
const int   mqtt_port     = xxx;             // порт для подключения к серверу MQTT  TLS: 3011
const char *mqtt_user     = "xxxx";       // логин от сервера MQTT
const char *mqtt_pass     = "xxxx";       // пароль от сервера MQTT

const char *led_topic     = "test/led";       // топик для светодиода
const char *data_topic    = "test/data";      // топик для данных


    const char *topic_gen_current = "current/g";  // топик публикации общ. значения тока
    // Топики для вкл./выкл. Стэндов
    const String stand[STANDS]=
    {
      "stand/1",  // Стэнд 1
      "stand/2",  // Стэнд 2
      "stand/3",  // Стэнд 3
      "stand/4",  // Стэнд 4
      "stand/5",  // Стэнд 5
      "stand/6",  // Стэнд 6
      "stand/7",  // Стэнд 7
      "stand/8",  // Стэнд 8
      "stand/9",  // Стэнд 9
      "stand/10", // Стэнд 10
      "stand/11", // Стэнд 11
      "stand/12"  // Стэнд 12
    };


      // Топики для публикации токопотребления
      const String topic_current[STANDS]=
      {
        "current/1",  // Ток стенда 1
        "current/2",  // Ток стенда 2
        "current/3",  // Ток стенда 3
        "current/4",  // Ток стенда 4
        "current/5",  // Ток стенда 5
        "current/6",  // Ток стенда 6
        "current/7",  // Ток стенда 7
        "current/8",  // Ток стенда 8
        "current/9",  // Ток стенда 9
        "current/10", // Ток стенда 10
        "current/11", // Ток стенда 11
        "current/12"  // Ток стенда 12
      };

      int pause = 300;                              // переменная для паузы между отправками данных
      long int times=0;                             // для времени

      WiFiClient wclient;
      PubSubClient client(wclient, mqtt_server, mqtt_port);
```

Arduino

```
// получение данных с сервера и отработка
void callback(const MQTT::Publish& pub)
{
    String topic = pub.topic();
    String payload = pub.payload_string();  // чтение данных из топика
    // действия над светодиодом в зависимости от данных из топика

    // Проверяем топики Стендов
    for (uint8_t i=0;i<STANDS;i++)
      if (topic==stand[i])
      {
        // Формируем байт для отправки команды
        uint8_t com = i+1;          // Номер стенда
        com*=10;                    // Смещаем номер слево
        if (payload[0]=='1')        // Состояние стенда   '1' - вкл.
          com+=1;

        Serial.write(com);          // Отправляем команду

        break;                      // Выход из цикла
      }
}
```

```
void data_UART()
{
  while(Serial.available())
  {
    digitalWrite(LED,LOW);                      // вкл. светодиод (пришли данные по UART)
    if (CountArr<BYTE_UART)                     // если еще не пришло нужное кол-во байт
        TX_Arr[CountArr++]=Serial.read();       // считывание байта
    else                                        // если набрали нужное кол-во
    {
      uint32_t data=0;                          // для формирования числа принятого по UART
      for (uint8_t i=0;i<BYTE_UART-1;i++)       // цикл, проходящий по всем эл. массива (кроме последнего)
        data|=(TX_Arr[i]<<(i*8));               // их полученных байт формируем число
      CountArr=0;                               // обнуление счетчика


      if ((char)TX_Arr[BYTE_UART-1]=='g')       // если последний байт G - общ. значение тока
        client.publish(topic_gen_current,String(data));// публикации общ. токопотребления
      else
        client.publish(topic_current[TX_Arr[BYTE_UART-1]],String(data));// ток стенд    (последний принятый байт номер стенда с 0)
      digitalWrite(LED,HIGH);                   // выкл. светодиод (данные считаны и обработаны)
    }
  }
}
```

Arduino

```
// Функция отправки показаний
void refreshData() {
  if (pause == 0) {
    times = millis(); // формируем данные для отправки
    client.publish(data_topic, String(times));
    pause = 3000; // пауза меду отправками 3 секунды
  }
  pause--;

  delay(1);
}
//-------------------------------------

void setup() {
    softSerial.begin(115200);       // Инициализация программного последовательного порта
    Serial.begin(115200);           // Инициализация порта

    pinMode(LED, OUTPUT);           // пин светика на выход
    digitalWrite(LED,HIGH);         // состояние светодиода выкл.

    // изначально токпотребление на всех стендах равно 0
    for (uint8_t i=0;i<STANDS;i++)
      Current_St[i]=0;
}


void subscription_topic() // ф-я подписки на топики
{

  for (uint8_t i=0;i<STANDS;i++)                // Подписка на топики Стэндов
    client.subscribe(stand[i]);

  for (uint8_t i=0;i<STANDS;i++)                // Подпись на топик публикации Покопотребления
    client.subscribe(topic_current[i]);

  client.subscribe(topic_gen_current);          // подписка на топик публикации общ. токопотребления

  client.subscribe(led_topic);                  // подписка на топик led
  client.subscribe(data_topic);                 // подписка на топик data
}


 void loop() {

    if (WiFi.status() != WL_CONNECTED) { // если соединения нет
        WiFi.begin(ssid, pass);// подключаемся к wi-fi
        if (WiFi.waitForConnectResult() != WL_CONNECTED) // ждем окончания подключения
            return;
    }

    // подключаемся к MQTT серверу
    if (WiFi.status() == WL_CONNECTED) { // если есть подключение к wi-fi
        if (!client.connected()) { // если нет подключения к серверу MQTTsetServer
           // Serial.println("MQTT - none");
            if (client.connect(MQTT::Connect(MQTT_client) // если соединились то делаем всякое
                               .set_auth(mqtt_user, mqtt_pass))) {
               // Serial.println("MQTT - ok");
                client.set_callback(callback);
                subscription_topic();             // подписка на топики
            }
            /*else { Вывод на дисплей
                Serial.println("MQTT - error");   // если не удалось подключиться сообщаем в порт
            }
            */
        }

        if (client.connected()){ // если есть соединение с MQTT
            client.loop();
            if (Serial.available())
                data_UART();
            //refreshData();
```