



Описание контроллера стиральной машины.

Контроллер предназначен для модернизации стиральных машин (СМ) с коллекторным двигателем имеющий индуктивный тахогенератор, двумя клапанами подачи воды, двухуровневым механическим датчиком уровня воды (прессостат), насосом слива воды, электрическим замком люка.

Форм фактор контроллера выполнен под размер EVO II. Контроль температуры нагрева воды осуществляется с помощью цифрового датчика температуры DS18b20. Логика работы устройства реализована на микроконтроллере SM32F103C8T6 который припаивается к печатной плате с обвязывающими его компонентами. Также, предусмотрена возможность установки отладочной платы «BluPill» вместо микроконтроллера.

Внешний вид платы контроллера.

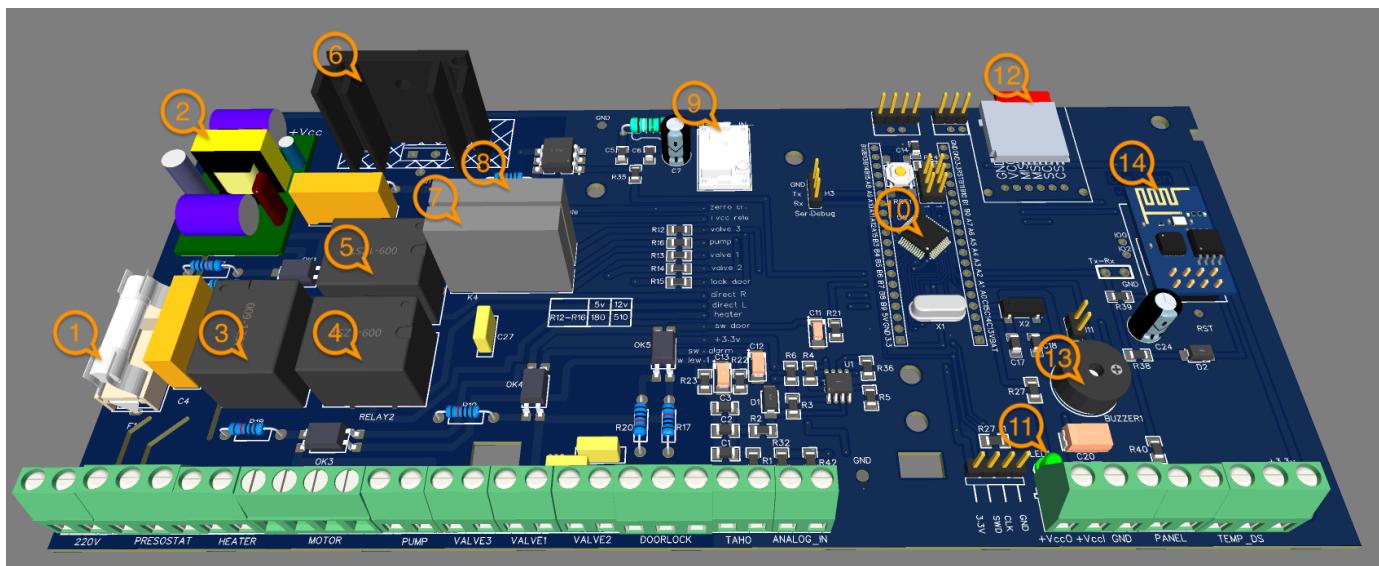


Рисунок 1. Электронная плата контроллера СМ, верхняя сторона, версия 1.

Основные компоненты контроллера:

1 – Плавкий предохранитель. **2** – Блок питания. **3** – Реле нагревателя. **4,5** – Реле реверса двигателя. **6** – Симистор управления двигателем на теплоотводе. **7** - Оптореле управлением насоса слива. **8** – Оптореле дополнительное (резерв). **9** – Импульсный понижающий преобразователь напряжения. **10** – Микроконтроллер SM32F103C8T6. **11** – Индикатор состояния контроллера – светодиод. **12** – Карт приёмник микро SD карты. **13** – Активный пьезоизлучатель. **14** – WIFI модуль ESP8266.

Основные возможности контроллера

- Количество поддерживаемых сценариев стирок – 255.
- Расположение программ стирок и настроек на внешнем носителе SD карте.
- Управление процессом стирки осуществляется с помощью сценариев и программ, написанных на скриптовом языке в текстовом формате, хранящихся в отдельных файлах для каждой программы.
- Управление контроллером по однопроводному последовательному интерфейсу.
- Управление контроллером по средствам WIFI.
- Автостарт стирки при отсутствии панели управления.
- Обновление программы контроллера с SD карты.
- Самотестирование и регистрация ошибок в файле при обнаружении.
- Блокировка включения нагрева воды при отсутствии в баке последней.
- Автоматический слив воды при переполнении бака.

*- не готово, тестируется...



Программы стирок – самостоятельные программы, состоящие из последовательных команд написанные на скриптовом языке. (Не путать с программами стирок на панели управления)

Сценарий стирки – последовательность выполняемых программ стирки, ассоциированный с панелью управления СМ.

Например: Стирка, Полоскание, Отжим – это отдельные три **программы стирок**, которые можно объединить в один **сценарий стирки**, который ассоциирован с надписью, значком на панели управления СМ.

Перечень команд используемых в программах стирок

Команда	Описание
V1;	включить подачу воды через первый клапан
V0;	выключить подачу воды первого клапана
W1;	включить подачу воды через второй клапан
W0;	выключить подачу воды второго клапана
J1;	включить дополнительный выход (например клапан, реле, насос)
J0;	выключить дополнительный выход (например клапан, реле, насос)
D1;	включить слив (насос) воды
D0;	выключить слив (насос) воды
S0; ... S199;	команда статуса стирки для панели управления. Статус может быть от 0 до 199. Статусы с 200 по 255 зарезервированы под сообщения системного характера, например, ошибок контроллера, сброса модуля, пароля WIFI.
B500;	включает звуковой сигнал (активный зуммер), например, 500 миллисекунд
B0;	выключает звуковой сигнал (зуммер)
T7;	воспроизводит аудио файл из директории WAV с именем 7.wav
L1;	включает блокиратор замка двери. Команда будет ждать выполнения блокировки. Пока не появится сигнал наличия блокировки.
L0;	выключает блокировку замка люка.
P120;	команда паузы в секундах. Пауза может быть от 1 до 65535 секунд. Используется для задержки выполнения команды после нее.
H35;	установка температуры воды в баке для стирки. H35; - нагревает воду до 35 °C. Команда будет ждать пока температура не достигнет 35 °C. После достижения заданной температуры, будет выполняться следующая команда.
H?;	знак вопроса (?) означает, что установка значения температуры берется с панели управления.
H0;	отключение поддержки температуры если такая имелась.
A1,1;	установка уровня воды на уровне 1 с помощью первого (V1) клапана. При установке уровня воды на 1-й отметке (A1,1;) - уровень будет автоматически поддерживаться.
A1,2;	установка уровня воды на уровне 1 с помощью второго (W1) клапана. При установке уровня воды на 1-й отметке (A1,2;) - уровень будет автоматически поддерживаться.



A3,1;	установить уровень воды на отметке 3 с помощью клапана 1. Т.е. будет налито воды три раза как уровень 1.
A0,1;	отключить контроль уровня воды и сброс значения память уровней используется при сливе воды из бака, второй параметр значения не имеет.

E20;	слив воды из бака до уровня воды ниже 1 с последующей паузой отключения насоса слива, например, 20 сек.
-------------	---

M35,7,6,10,5;	команда управлением привода барабана. 35 - кол-во оборотов в минуту, 7 - сек. вращать по часовой стрелке, 6 - шесть сек пауза после вращения почасовой стрелке, 10 - сек. вращение против часовой стрелки, 5 - сек. пауза после вращения против часовой стрелки и так далее по кругу. Выполнение команды не препятствует выполнению следующим командам. Скорость вращения может быть от 10 до 100 оборотов в мин. !!!Двигатель имеет инерцию, резкая смена направления вращения может привести к поломке двигателя и механизма привода барабана.
M0,0,0,0,0;	остановка и сброс параметров вращения двигателя.

O500,240,10,2;	отжим белья на 500 оборот в мин, в течении 240 сек, 10 - попыток балансировки барабана, 2 - компаратор дисбаланса Обороты от 150 до 1000, 240 сек (время отжима) - от 60 до 65535 сек, 10 попыток от 1 до 65535 (обычно 10), 2 компаратор от 1 до 9 (обычно 2 или 3), чем меньше, тем стабильнее.
O?,120,10,1;	отжим белья на скорости, заданной с панели управления. Знак вопроса (?) - использовать значение оборотов отжима с панели управления.

Пример скрипта программы стирки:

```
S1;P3;L1;V1;P30;V0; M30,10,5,12,7; P120;M0,0,0,0,0;D1;P30;D0;L0;S0;
```

Вышеприведенный скрипт практического применения не имеет. Он демонстрирует расположение команд в файле скрипта.

Команды в файле скрипта должны отделяться друг от друга точкой с запятой (;). Перед, внутри и после команды недолжно быть пробелов, табуляторов, переводов строк и т.п. Все команды начинаются с заглавной латинской буквы. Параметры внутри команды, если их несколько, должны иметь разделитель виде запятой (,).

Приведенный скрипт выполняет следующие действия:

1. **S1;** - отправить статус 1 для панели управления.
2. **P3;** - паза в 3 секунды до выполнения следующей команды;
3. **L1;** - блокировка замка люка СМ, с ожиданием сигнала закрытого люка. Следующая команда не будет выполняться до тех пор, пока не появится сигнал о том, что люк закрыт.
4. **V1;** - включить клапан подачи воды в СМ.
5. **P30;** - пауза 30 секунд. Выполнение следующей команды возможно только после 30 секунд.
6. **V0;** - выключить клапан подачи воды в СМ.
7. **M30,10,5,12,7;** - включить вращение барабана со скоростью 30 об/мин на 10 секунд в одну сторону с последующей остановкой на 5 сек и вращением барабана со скоростью 30 об/мин в противоположную сторону длительностью 12 сек и паузой в 7 сек.
8. **P120;** - пауза 120 сек.
9. **M0,0,0,0,0;** - выключить вращение барабана;
10. **D1;** - включить насос слива воды из бака.
11. **P30;** - пауза 30 сек.
12. **D0;** - выключить насос слива воды из бака.
13. **L0;** - отключить блокировку люка СМ.



14. **S0**; - отправить статус 0 для панели управления.

Для написания и отладки скриптов лучше всего воспользоваться специализированным редактором с подсветкой синтаксиса, расположенного в директории «**DOC**» SD карты. Также, редактор и отладчик доступен в сети Интернет по адресу: <https://oleston.github.io/>

Если перейти по адресу редактора в сети Интернет, скопировав или написав скрипт как выше в редактируемом поле редактора. Запустить скрипт с помощью зеленой кнопки с треугольником, то можно наблюдать процесс выполнения программы. (Для закрытия люка СМ, щелкните по контуру люка или установите/снимите галочку «Open / Close door:» справой стороны окна.)

Файловая структура SD карты

Micro SD карта должна иметь форматирование FAT32, один раздел, объемом равной или более 256 Мб.

Структура расположения директорий и файлов SD карты.

SDcard
LOG.LOG
ID.NUM
ARUN
R
DOC
EDITOR
MANUAL
FRMW
PROG.BIN
MOT
PID
REDU
ТАНО
PROG
0
1
2
...
...
254
255
STEP
0
1
2
...
...
254
255
WAV
0.wav
1.wav
2.wav
...
...
254.wav
255.wav
WIFI
ESP

Жирным шрифтом выделены директории. Остальное – файлы.

ARUN – содержит файл R для автоматического запуска программы стирки при подаче питания на микроконтроллер. Внутри файла R указывается номер сценария стирки, заканчивающийся точкой с запятой (;). 0; - автозапуск выключен. Больше 0 (ноля) – автозапуск включен, будет выполняться сценарий стирки, соответствующий числу в файле.

DOC – содержит документацию (**MANUAL**), редактор программ (**EDITOR**) и др.

FRMW – служит для обновления программы контроллера. При обнаружении в папке файл с именем PROG.BIN во время включения контроллера, начинается процесс обновления, с последующим переименованием файла на PROG.BI_

MOT – хранит конфигурационные файлы мотора. Файл PID отвечает за настройки ПИД регулятора, содержит три параметра разделенные точкой с запятой (;). Первый параметр Р, второй I, третий D. Пример содержания файла: 45.0;1.5;0.0; Файл REDU – отвечает за параметр соотношение оборотов вала двигателя к оборотам барабана. Один параметр, например: 14; Четырнадцать оборотов вала двигателя к одному обороту барабана. Файл ТАНО – содержит параметр, отвечающий за кол-во импульсов, выдаваемых тахогенератором двигателя за один оборот, например: 8;

PROG – директория с файлами программ стирок. Допускается использовать только цифровые номера.

STEP – директория с файлами сценариев стирок. Файлы должны содержать перечень действительных файлов программ стирок, разделенные точкой с запятой (;). Пример: 4;7;13; или 9;

WAV – содержит звуковые файлы, формата wav, для проигрывания информационных звуков, сообщений. Имена файлов должны быть числовыми, с расширением wav. Пример: 14.wav

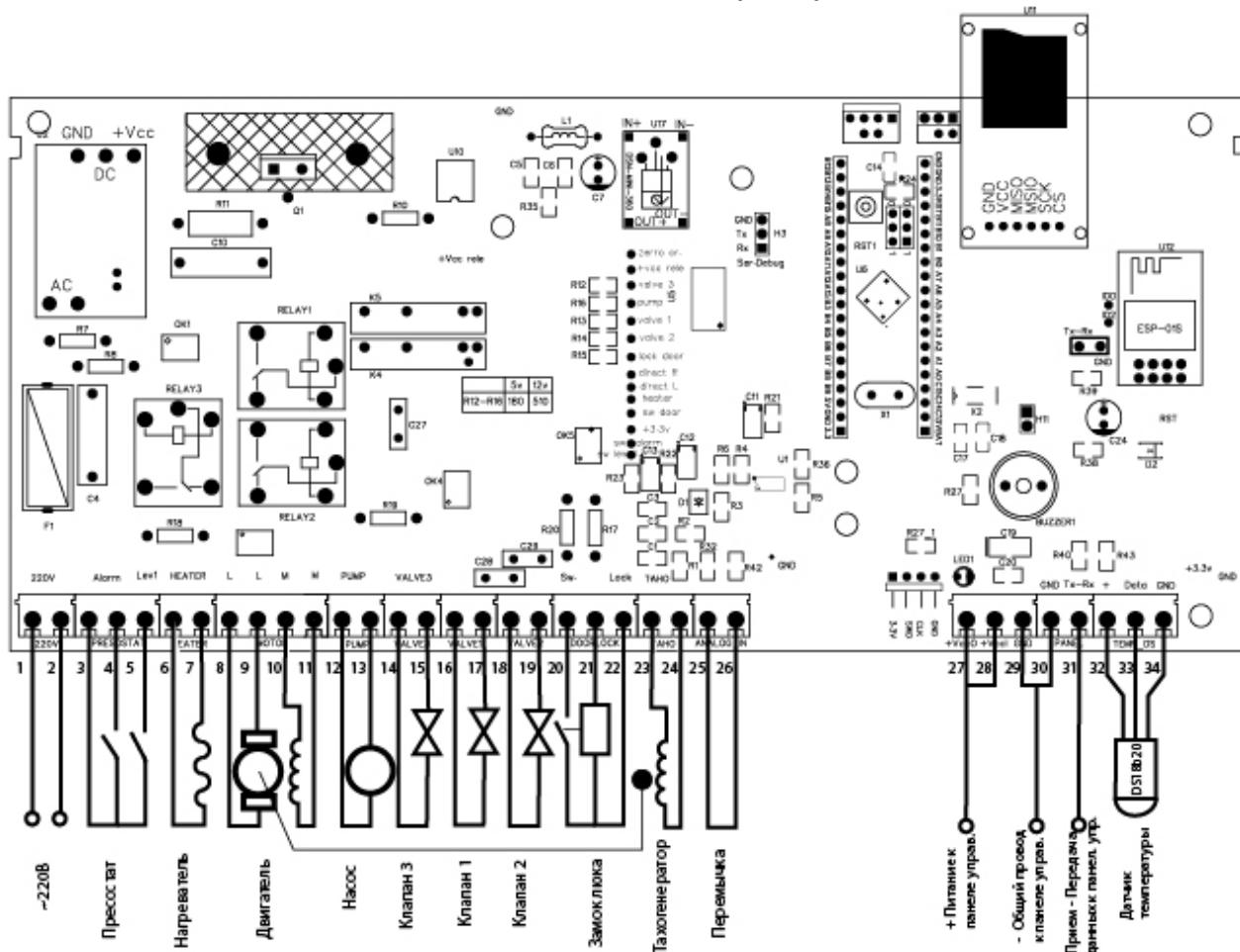
WIFI – хранит конфигурационный файл ESP отвечающий за использование wifi панели управления. Содержит один параметр 0; - выключено, 1; - включено.

LOG.LOG – файл журнал в который вносятся ошибки и др. информация от микроконтроллера.

ID.NUM – файл с идентификатором микроконтроллера. Необходим для обновления программы.



Схема подключения контроллера

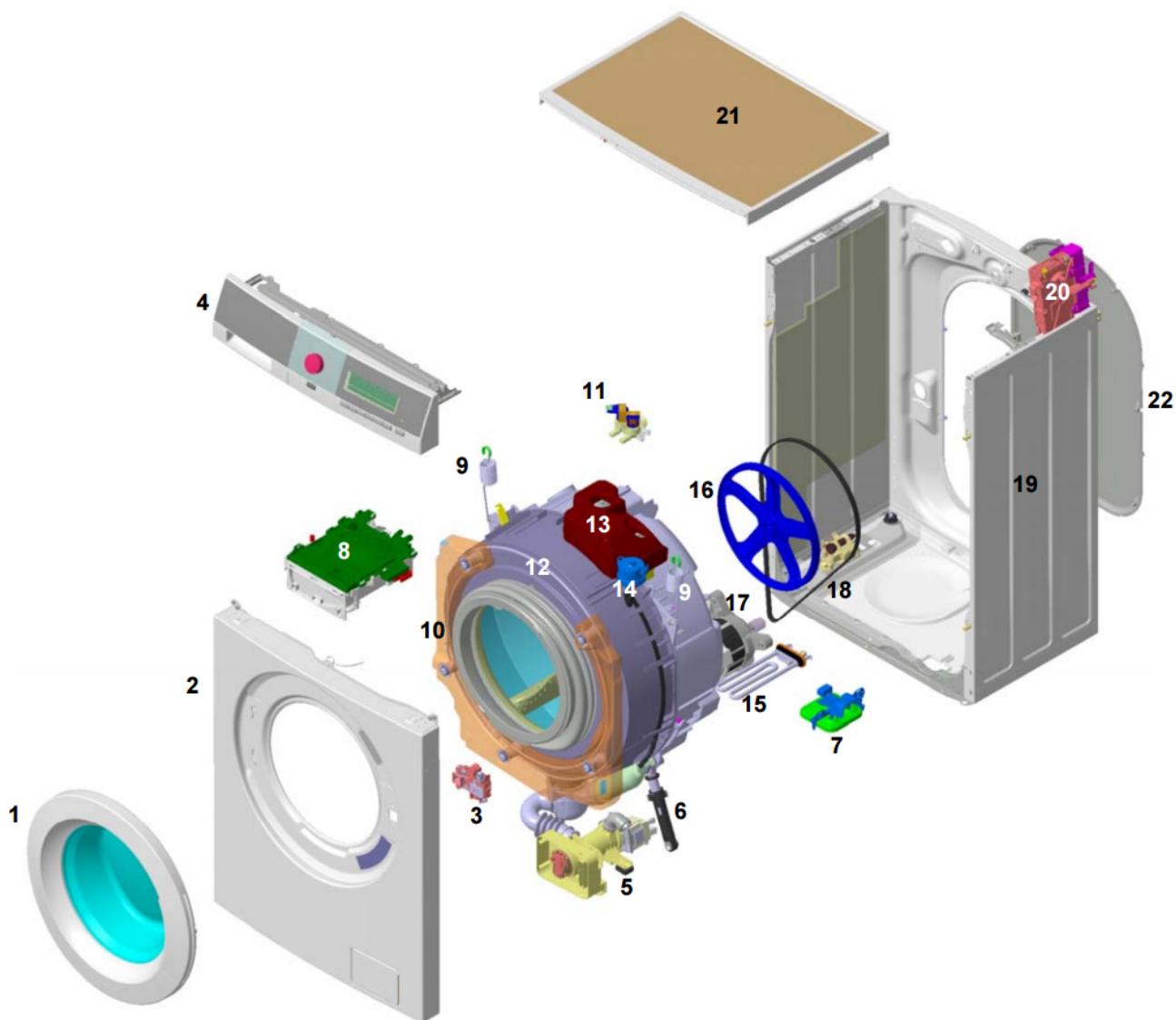


Контакт	Исполнительное устройство СМ.
1,2	Подключение питания устройства от сети переменного напряжения ~220В.
3,4,5	Датчик уровня воды в баке (прессостат). 4 – перелив. 5 – уровень 1.
6,7	Нагреватель воды (ТЭН).
8,9	Обмотка якоря двигателя привода барабана.
10,11	Обмотка статора двигателя привода барабана.
12,13	Насос слива воды из бака (помпа).
14,15	Клапан – резерв.
16,17	Клапан подачи воды - 1.
18,19	Клапан подачи воды - 2.
20,21,22	Замок люка СМ.
23,24	Обмотка таходатчика двигателя привода барабана.
25,26	Перемычка – резерв.
27	Выход (плюс) питания панели управления и др. устройств.
28	Вход (плюс) питания логического узла микроконтроллера.
29,31	Общий провод питания контроллера для питания панели управления и др. устройств.
32,33,34	Датчик температуры воды DS18b20. 32 – плюс 3.3В. 33 – Данные. 34 – Общий провод питания датчика температуры.



Конструкция, расположение основных узлов стиральной машины.

(Иллюстрация взята из каталога на стиральную машину ELECTROLUX)



1 - Люк стиральной машины. **2** - Передняя панель корпуса стиральной машины. **3** - Устройство блокировки люка (УБЛ), блокиратор, замок люка. **4** - Панель управления. **5** - Сливной насос, помпа в сборе с корпусом и фильтром- заглушкой. **6** - Амортизаторы, направляющие, 2 шт. **7** - Аквастоп (устанавливается не на все стиральные машины). **8** - Диспенсер, дозатор моющего средства, бункер для порошка. **9** - Пружины бака, 2 шт. **10** - Манжета люка, уплотнитель резиновый. **11** - Заливной электромагнитный клапан, соленоид. **12** - Бак. Внутри бака находится барабан, который снабжен активаторами стирки. **13** - Противовес. **14** - Прессостат, датчик уровня, датчик давления. **15** - Нагревательный элемент, ТЭН. **16** - Шкив. **17** - Двигатель, мотор стиральной машины. **18** - Ремень привода барабана. **19** - Корпус стиральной машины. **20** - Модуль управления. Не всегда расположен так, как показано на картинке. **21** - Верхняя крышка. **22** - Задняя стенка. **23** - Патрубок.

Подключение исполнительных устройств к контроллеру СМ.

Монтаж исполнительных устройств осуществляется путем подсоединений токоведущих проводников к соответствующим клеммным винтовым колодкам.

Датчик температуры DS18b20, устанавливается вместо штатного, использовав заглушку 87885 (см. картинку ниже), с последующим приклеиванием сенсора теплопроводящим kleem. При отсутствии заглушки 87885, можно использовать корпус штатного датчика температуры, предварительно извлечь последний.



Освободить корпус от штатного сенсора легче всего, путем нагрева металлической оболочки его и вытягиванием внутренностей.

Датчик температуры.



Заглушка датчика температуры 87885.



В большинстве случаев датчики температуры воды имеют конструкцию как выше и установлены на площадку крепления нагревателя воды (ТЭН).



Для правильного подключения к контроллеру СМ коллекторного двигателя, который обеспечивает вращение барабана, необходимо безошибочно определить его контактные пары обмоток статора, ротора и тахогенератора. Обычно, двигатель расположен в нижней части стиральной машины, закрепленный на баке.



Типичный общий вид коллекторного двигателя представлен на картинке слева.

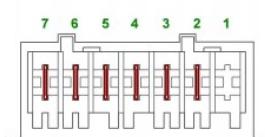
С левой стороны находится клеммная колодка, с тыльной стороны, на валу тахогенератор. Двигатели разных производителей имеют отличия в части очередности токопроводящих выводов клеммной колодки. Для определения пар понадобится мульти метр, с возможностью определения сопротивления цепей или документация на двигатель. Также, можно оценить визуально куда идут на какие клеммы провода.

Сопротивление тахогенератора имеет повышенное относительно обмоток статора и ротора.

Ниже представлены распиновки клеммных колодок часто встречающихся коллекторных двигателей.
(Источник: <https://sw19.ru>)

Whirlpool AWE6377

IBA5736-OWA 230/240V 50Hz Iso.K1.F
461975025911 054/07 2
20/01 1139



1 - x

2,3 - статор

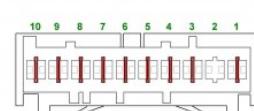
4,5 - ротор

6,7 - тахо (13.8 Ом)

Ingenyuu.info

LG EMERSON

U.M. MCC52/64 - 148/LG2
12000rpm - 1.5A - 340W 220 Vdc - cl F/F
Thermally protected
LG P/N 4681EN1010D
C.E.SET
EMERSON(CHINA)MOTOR CO., LTD



1,5 - статор (I)

1,10 - статор (II)

2 - x

3,4 - тахо (64,1 Ом)

6,7 - термопредохранитель

8,9 - ротор

1 - оранжевый

3,4 - красный

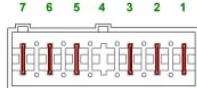
5 - чёрный

6,7 - голубой

8 - розовый

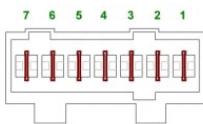
9 - синий

10 - серый Ingenyuu.info



- 1,2 - ротор
- 3,5 - статор
- 4 - x
- 6,7 - тахо (74.8 Ом)
- 1 - чёрный
- 2 - серый
- 3 - красный
- 5 - коричневый
- 6,7 - красный

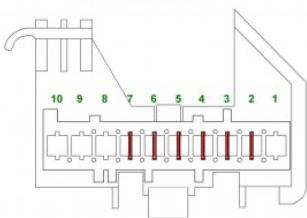
Gorenje
C.E.SET.
Motore universale
MCA 52/64-148/KT11
13000 rpm - 1,7 A - 390 W
230-240 V - 50 Hz - cl F/F
Thermally protected
Rif. 163959



- 1,2 - статор I (0,9 Ом)
- 1,3 - статор II (0,8 Ом)
- 4,5 - ротор
- 6,7 - тахо (67.3 Ом)
- 1 - оранжевый
- 2 - коричневый
- 3 - чёрный
- 4 - розовый
- 5 - синий
- 6,7 - красный

Electrolux, Candy
ACC

9844347.0
Type20580.052 AC-EL CI B/F
230/240V 50Hz 480/14000 RPM
132529611
MU1-1 P616
6-38-E .CO. 07.10



- 1,8,9,10 - x
- 2,5 - статор
- 3,4 - ротор
- 6,7 - тахо (180 Ом)
- 2 - салатовый
- 3 - зелёный
- 4 - белый
- 5 - коричневый
- 6,7 - синий

Тахогенераторы имеют разные выходное параметры, в части количества импульсов на один полный оборот шпинделя двигателя. Для определения кол-ва импульсов необходимо обратиться к документации на данный вид двигателя. Также, на внутренней части крышки таходатчика имеется маркировка о его параметрах.



Фотография крышки таходатчика слева.

На внутренней части крышки датчика присутствует маркировка указывающая, что генератор выдает 8 импульсов на один оборот вала.
Стрелка указывает на кол-во импульсов.

Датчик уровня воды (прессостат) имеют различные виды исполнения и отличия в контактах группах на своем корпусе. Классический прессостат состоит минимум из двух включателей, включенных параллельно имеющие общий вывод. Одна контактная пара отвечает за первый уровень воды, при котором возможен нагрев и основной цикл стирки с поддержанием уровня жидкости. Вторая, контактная пара служит для аварийного слива воды при переполнении бака. Датчик уровня воды крепится в верхней точке стиральной машины, имеет отвод из трубки к нижней части бака СМ.

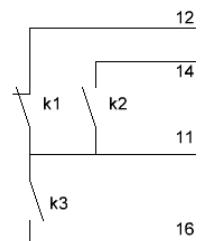


На фотографии слева одна из часто встречающихся конструкций механического датчика уровня воды (прессостат). В нижней части реле находится отвод для подключения гибкого, резинового шланга соединенного с нижней частью бака СМ. На прессостате имеется клеммная колодка, регулировочные винты уровня срабатывания контактных пар. Некоторые конструкции механических датчиков уровней воды имеют более двух контактных пар, которые могут работать как на замыкание, так и на размыкание. Для текущего контроллера МС необходимо использовать два переключателя работающих на замыкание при достижении определенного уровня воды.

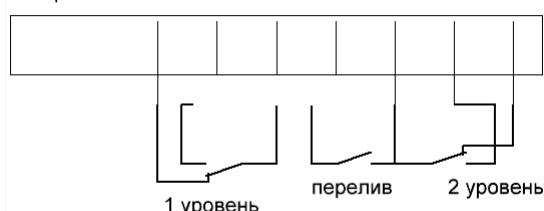
Для определения необходимых контактов нужно воспользоваться измерительным прибором с возможностью прозвонки электрических цепей, типа мульти метр. Демонтированный прессостат, резиновая трубка, одетая на устройство и пластиковая бутылка с отрезанным дном. К пробке бутылки, через проделанное отверстие необходимо подключить свободный конец трубы от реле уровня воды. Вертикально помещая бутылку отрезанным дном в емкость с водой, будет наблюдаться звук срабатывания контактных групп прессостата, в зависимости от глубины погружения. При срабатывании одной из групп, необходимо выяснить какие контакты реле замкнулись.

Пример внутренних схем двух производителей датчиков уровней воды.

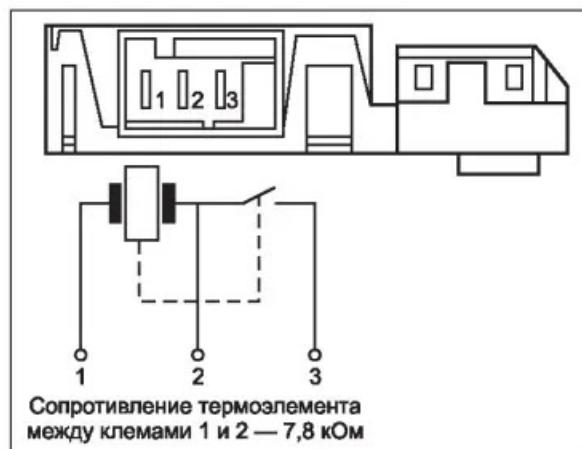
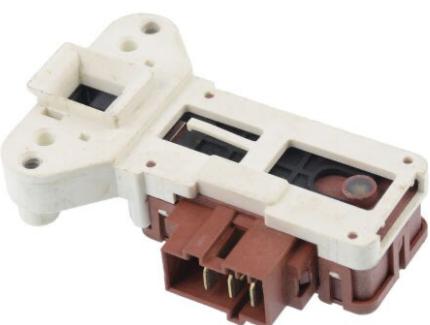
Прессостат Ariston Indesit



Прессостат BOSCH



Замок люка СМ (УБЛ) имеют разные конструкции. Но подавляющая большинство их трехконтактная, состоящая из нагревательного элемента и контактной пары. Вид и схема замка представлены ниже.



Остальные элементы, такие как насос слива воды, клапаны, нагреватель имеют по два вывода и сложность в подключении их не составит особого труда к предназначенным местам клеммных колодок контроллера СМ.



Интерфейс связи с панелью управления.

Последовательный однопроводный интерфейс:

- полудуплексный режим.
- скорость передачи данных – 9600 бит/сек.
- данные – 8 бит.
- стоп бит – 1.
- паритет – нет.
- амплитуда – 3.3В

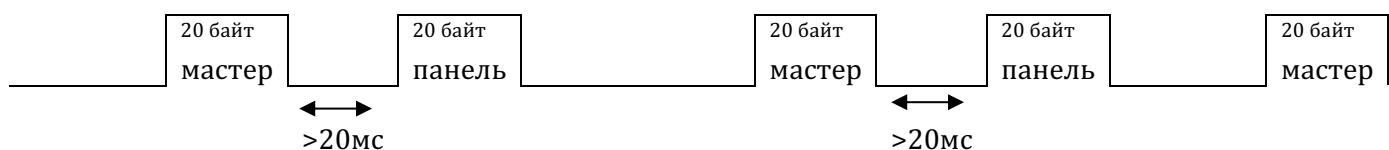
Формат обмениваемыми данными с панелью управления.

Кадр передаваемых и принимаемых данных состоит из 20 байт. Инициатором обмена данных выступает контроллер (мастер) СМ, отправляя данные по 20 байт с последующим ожиданием приема данных от панели управления. Передача данных от контроллера происходит, каждые 500 миллисекунд. После отправки данных, контроллер (мастер) СМ переходит в режим ожидания приема данных до следующей передачи. Отвечающая сторона, панель управления, после приема данных должна отправить ответ контроллеру не ранее чем через 20 миллисекунд. Ответ должен состоять из такого же количества данных, т.е. из 20 байт.

Кадр передаваемых, принимаемых данных.



Диаграмма очередности приема передачи данных.



Байт	Назначение	Направление
0	Идентификатор (ID) для кого сообщение. 1 – Мастер. 2 – Панель. 3 - Wi-Fi.	В/из Контроллер
1	Старт стирка – 1. Стоп стирка – 0.	В Контроллер
2	Сценарий стирки от 0 до 255.	В Контроллер
3	Температура стирки от 0 до 95. Если 0, то без нагрева воды.	В Контроллер
4	Обороты отжима от 0 до 100. В контроллере умножается на 10. Если 0, то без отжима.	В Контроллер
5	Статус стирки для обозначения состояния стирки на панели управления.	Из Контроллера
6	Флаг от контроллера. 1 – конец стирки. 0 – контроллер работает.	Из Контроллера
7	Зарезервировано.	
8	Зарезервировано.	
9	Текущая температура воды.	Из Контроллера
10	Заданная температура воды.	Из Контроллера
11	Уровень воды. 0 – меньше уровня 1. 1 – уровень 1. 2 – переполнение бака.	Из Контроллера
12	Номер исполняемой программы стирки.	Из Контроллера
13	Флаг состояния люка. 0 – открыт. 1- закрыт.	Из Контроллера
14	Кол-во сценариев стирки на SD карте в каталоге STEP.	Из Контроллера
15	Старший байт текущих оборотов барабана.	Из Контроллера
16	Младший байт текущих оборотов барабана.	Из Контроллера
17	Старший байт значения паузы.	Из Контроллера



18	Младший байт значения паузы.	Из Контроллера
19	Контрольная сумма (CRC).	В/из Контроллер



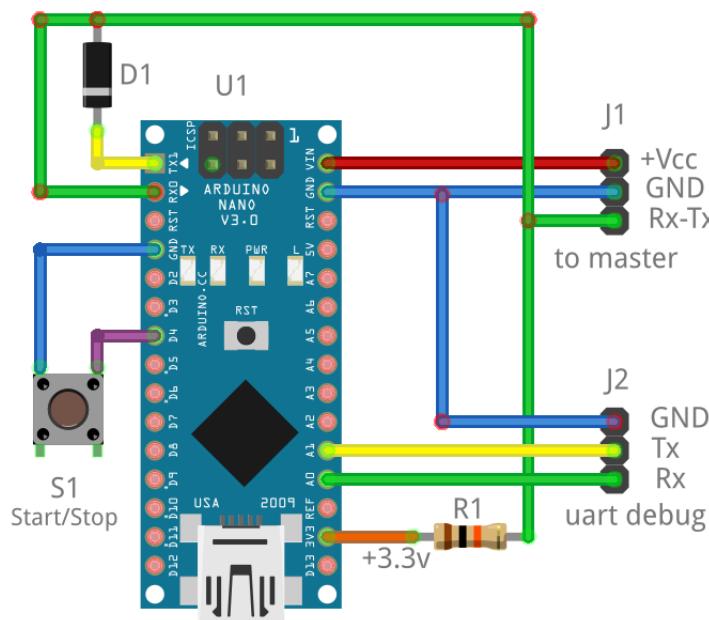
Панели управления контроллером стиральной машины.

Из-за многообразия конструкций панелей управлений различных СМ рассмотрим некоторые возможные реализации панелей управлений контроллером OleSton.

Пример №1.

Управление контроллером СМ по однопроводному интерфейсу с помощью Arduino Nano.

Упрощенная модель панели управления на базе Arduino Nano.



На приведенном рисунке слева представлена упрощённая схема панели управления контроллером СМ.

Кнопка **S1** отвечает за запуск и остановку стирки. Контакт **Tx** контроллера Arduino Nano соединен с контактом **Rx** через диод «Шоттки» **D1**.

Резистор **R1** 10kOhm подтягивающий к плюсу 3.3v.

Разъем J1:

+Vcc - питание Arduino Nano **U1**.

GND - общий провод между панелью управлением и контроллером СМ.

Rx-Tx - контакт входа выхода данных от/к контроллеру СМ.

Разъем J2* (опционально):

GND - общий провод.

Tx - контакт выхода отладочной информации контроллера панели управления.

*- для вывода отладочной информации в терминал ПК необходимо воспользоваться конвертором типа USB-TTL-PL2303HX

Пример 1 - программы для среды Arduino IDE.

Листинг программы, примера 1, панели управления. Среда Arduino IDE.

```
#include <SoftwareSerial.h>

#define PpanelID    2 // номер панели СМ в сети
#define MasterID    1 // номер контроллера СМ в сети

#define ProgNuber   8 // номер сценария стирки
#define TemperWash 30 // температура воды стирки
#define SpeedSpin   40 // скорость отжима, на контрол. СМ будет умножена на 10

#define LedPin      13 // номер пина светодиода
#define ButtPin     7 // номер пина кнопки
bool flag_bt = false; // флаг блок.дребез. кнопки
bool flag_send = false; // флаг режима отпраки данных контроллеру
uint8_t errConn = 0; // счетчик ошибок приема данных от контроллера
uint32_t btnTimer = 0; // переменная хранения миллисекунд
uint8_t StartWash = 0; // переменная хранения режим Стирки (Старт или Стоп)
void ButtonRead(void); // функция обработки состояния кнопки Старт/Стоп
SoftwareSerial softSerial(A4, A5); // Создаём объект SoftSerial указывая выводы
//RX, TX (можно указывать любые выводы Arduino UNO)
```



```

struct Str {
    uint8_t ID;           // id номер для кого сообщение
    uint8_t Start;        // Старт / Стоп стирка
    uint8_t Prog;         // Номер сценария стирки
    uint8_t Temper;       // Температура воды стирки
    uint8_t Speen;        // Обороты отжима белья
    uint8_t Status;       // Статус стирки
    uint8_t FlagEnd;     // Флаг окончания стирки
    uint8_t rez1;         // резерв
    uint8_t rez2;         // резерв
    uint8_t cTemp;        // Текущая температура воды в баке
    uint8_t sTemp;        // Заданная температура воды
    uint8_t LevWat;       // Уровень воды в баке
    uint8_t NumProg;      // Текущая исполняемая прогр. стирки из сценария стирки
    uint8_t FlagDoor;     // Флаг состояния люка
    uint8_t CountProg;    // Количество сценариев стирок на SD карте
    uint16_t Speed;       // Текущая скорость вращения барабана
    uint16_t Pause;       // Текущее значение паузы
    uint8_t CRC;          // резерв
};

Str buf; // создаём структуру для приема и отправки данных от/к контроллеру СМ

void setup() {
    pinMode(ButtPin, INPUT_PULLUP);           // настраиваем pin кнопки на вход с подтяжкой
    pinMode(LedPin, OUTPUT);                 // настраиваем pin светодиода на выход

    softSerial.begin(115200);                // Инициируем передачу данных по программной
                                            //шине UART на скорости 11520 (между Arduino и компьютером)

    softSerial.write("\nHello!\n");
    Serial.begin(9600);                     // Инициируем передачу данных по аппаратной шине UART
                                            //на скорости 9600 (между Arduino и модулем)
}

void loop() {
    ButtonRead();                          // читаем состояние кнопки Старт/Стоп
    sendData();                            // отправка данных мастеру
    checkErr();                            // Если от мастера выставлен флаг конец стирки
    {
        StartWash = 0;                    // Отключаем стирку на панели
        buf.FlagEnd = 0;                 // Очищаем флаг конца стирки
        buf.Start = 0;
    }
}

void ButtonRead()                      // функция чтения состояния кнопки с подавлением дребезга
{
    bool btnState = !digitalRead(ButtPin); // читаем инвертированное значение
    if (btnState && !flag_bt && millis() - btnTimer > 100)
    {
        StartWash ^= 1;                // инвертируем переменную при каждом нажатии кнопки
        digitalWrite(LedPin, StartWash); // отображаем состояние Старт/Стоп светодиодом
        flag_bt = true;              // переключаем флаг для запрета повторного входа в функцию
        btnTimer = millis();           // запоминаем значение миллисекунд
        softSerial.print("\nStartWash: ");
        softSerial.println(StartWash);   // для отладки выводим состояние переменной
    }
    if (!btnState && flag_bt && millis() - btnTimer > 100)
    {
        flag_bt = false;            // по истечении времени опускаем флаг
        btnTimer = millis();           // запоминаем значение миллисекунд
    }
}

void serialEvent()                   // функция получения символа в UART
{
    if (Serial.available() > (byte)sizeof(buf)) // Если пришло сообщение размером с нашу структуру
    {
        byte tmp[(byte)sizeof(buf)] = {0,};
        Serial.readBytes((byte*)&tmp, (byte)sizeof(buf));
        byte crc = 0;
        for (byte i = 0; i < 20 - 1; i++)
        {
            crc ^= tmp[i];
        }
        crc ^= 20 + 2;
        if (tmp[(byte)sizeof(buf) - 1] != crc || tmp[0] != PpanelID)
        {
            errConn++;
        }
    }
}

```



```

        softSerial.print("  ERR ");softSerial.print(errConn);
        return;
    }

memcpy((byte*)&buf, tmp, sizeof(tmp));

softSerial.print("\n");
softSerial.print(buf.ID);softSerial.print(" ");softSerial.print(buf.Start);
softSerial.print("softSerial.print(buf.Prog);softSerial.print(" ");
softSerial.print(buf.Temper);softSerial.print(" ");.print(buf.Speen);
softSerial.print(" ");softSerial.print(buf.Status);softSerial.print(" ");
softSerial.print(buf.FlagEnd);softSerial.print(" ");
softSerial.print(buf.rez1);softSerial.print(" ");softSerial.print(buf.rez2);
softSerial.print(" ");softSerial.print(buf.cTemp);softSerial.print(" ");
softSerial.print(buf.sTemp);softSerial.print(" ");
softSerial.print(buf.LevWat); softSerial.print(" ");
softSerial.print(buf.NumProg);softSerial.print(" ");
softSerial.print(buf.FlagDoor);softSerial.print(" ");
softSerial.print(buf.CountProg); softSerial.print(" ");
softSerial.print(buf.Speed); softSerial.print(" ");
softSerial.print(buf.Pause); softSerial.print(" ");
softSerial.println(buf.CRC);
softSerial.print("\n");

    if (buf.FlagEnd == 1)                                // Если от мастера выставлен флаг конец стирки
    {
        StartWash = 0;                                  // Отключаем стирку на панели
        buf.FlagEnd = 0;                                // Очищаем флаг конца стирки
    }
    flag_send = true;                                 // Поднимаем флаг отправки данных
    delay(50);                                       // Задержка перед отправкой данных
}

void sendData()
{
    if (flag_send == false) return;                  // Если флаг не поднят выходим из функции
                                                    // отправки данных

    byte tmp_buf[20] = {0,};
    tmp_buf[0] = MasterID;
    tmp_buf[1] = StartWash;
    tmp_buf[2] = 10;
    tmp_buf[3] = 40;
    tmp_buf[4] = 50;
    for (byte i = 0; i < 20 - 1; i++)           // Подсчет CRC отправляемых данных
        tmp_buf[19] ^= tmp_buf[i];
    buf.CRC = tmp_buf[19] ^ 20 + 2;

    Serial.write((byte*)&tmp_buf, sizeof(tmp_buf)); // Отвечаем мастеру
    flag_send = false;
}

void checkErr()                                     // Функция инициализации послед. порта
{
    if (errConn < 1) return;
    errConn = 0;
    Serial.end();
    Serial.begin(9600);
}

```

Приведенный выше листинг скетча программы для Arduino Nano очень прост. Основа программы лежит в приеме данных от контроллера (мастера) СМ с помощью последовательного интерфейса. Отладочная информация, при необходимости, отправляется в виртуальный последовательный интерфейс, реализованный с помощью библиотеки SoftwareSerial. При поступлении данных от управляющего контроллера СМ (мастера), возникает аппаратное прерывание контроллера панели управления (Arduino Nano) перенаправляемое в процедуру обработки **void serialEvent()**.

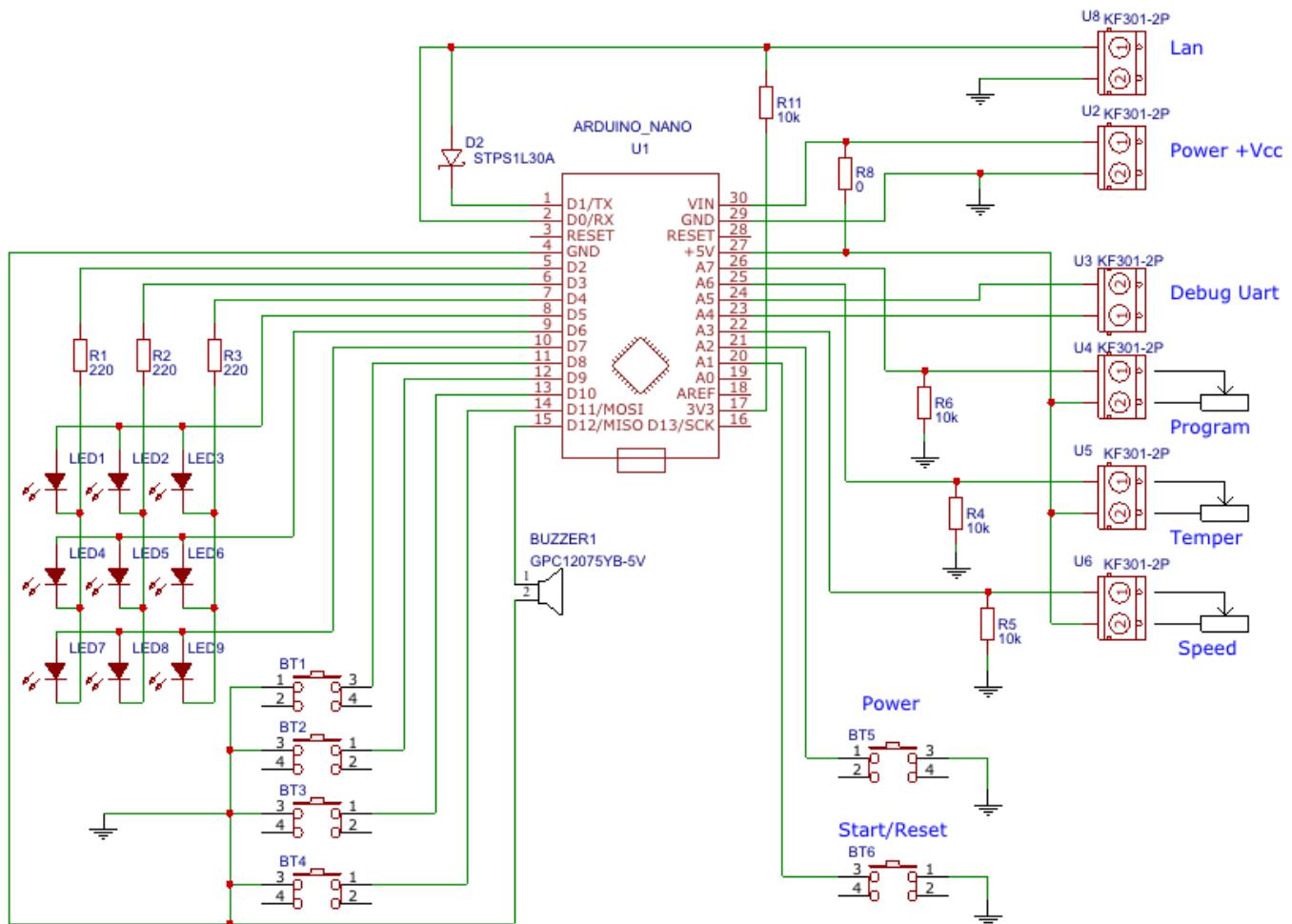
Из недостатков такой реализации необходимо отметить, что существует постоянная потребность в контроле состояния кнопки без использования прерываний микроконтроллера Arduino, также недостаточная информативность состояния контроллера и отсутствие органов выбора режимов стирки. Но простота кода, с учетом недостатков, наглядно демонстрирует алгоритм управления контроллером СМ с помощью однопроводного последовательного интерфейса.



Пример №2.

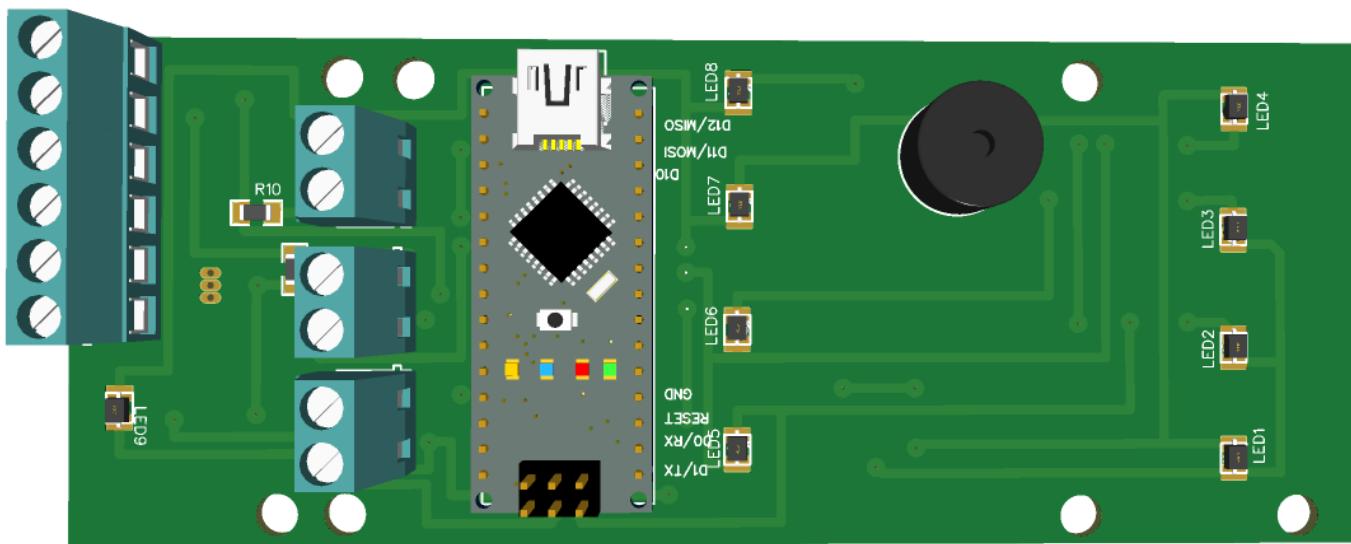
Управление контроллером CM (Ariston AVSL 100) по однопроводному интерфейсу на базе Arduino Nano, более приближен к оригинальному (заводскому) управлению контроллером CM, имеет более расширенный функционал.

Принципиальная схема контроллера панели управления CM (Ariston AVSL 100) на базе Arduino Nano.

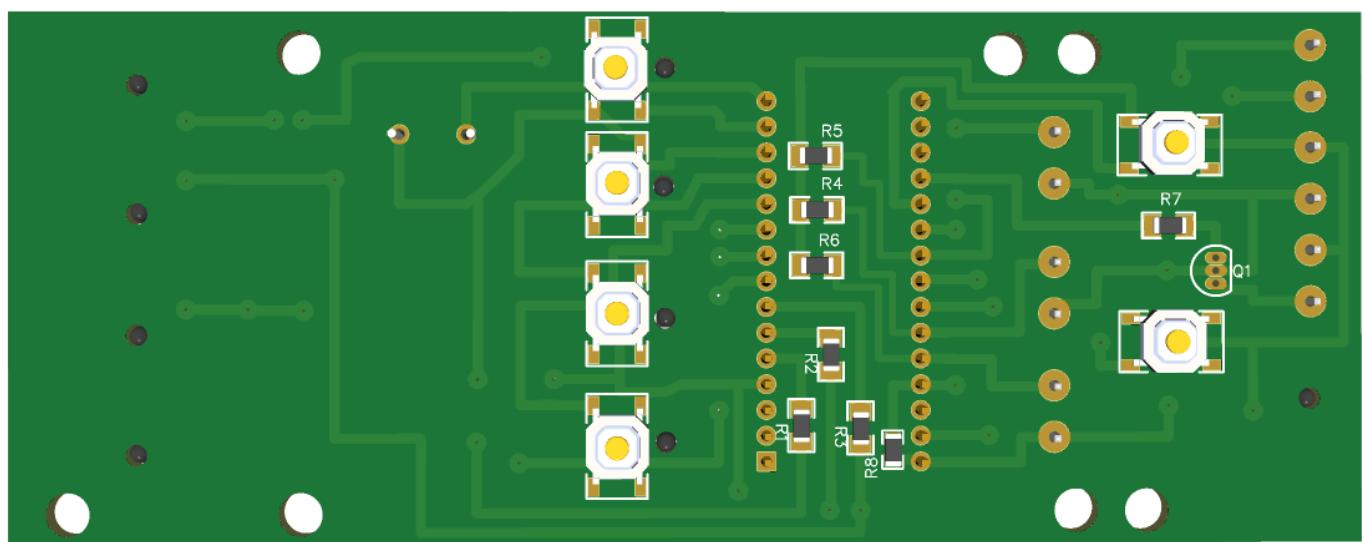




Макет* печатной платы контроллера панели управления CM (Ariston AVSL 100) на базе Arduino Nano.
Размер печатной платы изготовлен под размер штампной платы контроллера управления CM.
Верхняя сторона.



Макет печатной платы контроллера панели управления CM (Ariston AVSL 100) на базе Arduino Nano.
Нижняя сторона.



* - на печатной плате отсутствуют часть компонентов. Приведено в качестве примера.

Листинг программы, примера 2, контроллера панели управления CM (Ariston AVSL 100) для среды Arduino IDE.

```
//Скетч панели управления на контроллере Arduino Nano
#define DEBUG 0
#if(DEBUG)
#include <SoftwareSerial.h>
SoftwareSerial softSerial(A5, A4); // Создаём объект SoftSerial указывая выводы RX(A5),TX(A4)
#endif

// anodes 2,3,4 cathodes 5,6,7

#define ON 1
#define OFF 0
#define h9 0
#define h6 1
#define h4 2
#define h2 3
```



```

#define sleep          4
#define power         10
#define runRes        9
byte data_C[13] = { // массив элементов управления панели
    0, // предвар. стирка      0 led
    0, // стирка                1 led
    0, // полоскание            2 led
    0, // отжим                  3 led
    0, // таймер отсрочки стирки 4 led key
    0, // супер стирка           5 led key
    0, // быстрая стирка         6 led key
    0, // дополнит полоскание    7 led key
    0, // Старт / Стоп            8 led
    0, // Старт / Стоп            9 key
    0, // Вкл / Выкл              10 key
    0, // резерв                 11
    0 // резерв                 12
};

const byte row_col[9] = {52, 53, 54, 62, 63, 64, 72, 73, 74}; // пины светодиодов anode-catode

#define buttNum 7
const byte buttonArr [7] = {8, 9, 10, 11, 15, 16, 14}; // пины кнопок и др.
bool toggle_LED13 = 0;

#define selectorTemp        A7
#define selectorSpeed       A6
#define selectorProg        A3

#define SwPower             A0
#define Beep                12
#define keyBeepTime         50
#define progNum             16
const int selectorProg_values[2][16] = {
    // положение регулятора считывание с аналог. входа
    {3, 90, 150, 220, 300, 370, 430, 500, 580, 630, 700, 780, 840, 920, 990, 1550},
    // программы стирки // программы стирки
    {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16} };

#define tempSpeedNum 8
const int selectorTempSpeed_values[3][8] = {
    // положение регулятора считывание с аналог. входа
    {10, 210, 460, 600, 750, 850, 960, 1550},
    // температура стирки
    {0, 30, 40, 50, 60, 70, 80, 90},
    // обороты отжима
    {70, 80, 90, 100, 0, 40, 50, 60} };

#define ID                 0
#define Start              1
#define Prog               2
#define Temper             3
#define Spin               4
#define Status             5
#define FlagEnd            6
#define rez1               7
#define rez2               8
#define cTemp              9
#define sTemp              10
#define LevWat             11
#define NumProg            12
#define FlagDoor            13
#define CountProg           14
#define SpeedH              15
#define SpeedL              16
#define PauseH              17
#define PauseL              18
#define CRC                19

byte Command_Massive[20] = { 0, // 0 ID
                            0, // 1 Старт стирка -1 Стоп стирка -0
                            0, // 2 Номер сценария стирки
                            0, // 3 Температура воды стирки
                            0, // 4 Обороты отжима белья
                            0, // 5 Статус стирки
                            0, // 6 Флаг окончания стирки
                            0, // 7 резерв
                            0, // 8 резерв
                            0, // 9 Текущая температура воды в баке
                            0, // 10 Заданная температура воды
                            0, // 11 Уровень воды в баке

```



```

0, // 12 Текущая исполняемая программа стирки из сценария стирки
0, // 13 Флаг состояния люка
0, // 14 Количество сценариев строк на SD карте
0, // 15 Текущая скорость вращения барабана Н
0, // 16 Текущая скорость вращения барабана Л
0, // 17 Текущее значение паузы Н
0, // 18 Текущее значение паузы Л
0 // 19 CRC
};

uint8_t errConn = 0; // счетчик ошибок приема данных от контроллера
bool flag_send = false; // флаг режима отправки данных контроллеру
unsigned long delay_send = 0; // переменная паузы отправки данных
int delayStart = 0; // счетчик отсрочки начала стирки
unsigned long longlastTime = 0; // переменная хранения счетчика для звук сигнала

#define ControlPanelID 2 // номер панели управления в сети
#define MasterID 1 // номер контроллера СМ в сети
/*=====
void setup()
{
#if(DEBUG)
softSerial.begin(115200); // Инициируем передачу данных по программной шине UART на скорости 11520
// (между Arduino и компьютером)
softSerial.write("\n\n\nHello! I'm panel controll...\n");
#endif
Serial.begin(9600); // Инициируем передачу данных по аппаратной шине UART на скорости 9600
//(между Arduino и мастером)
pinMode(selectorTemp, INPUT );
pinMode(selectorSpeed, INPUT );
pinMode(selectorProg, INPUT );
for (int i = 0; i < 9; i++)
{
  pinMode(row_col[i] / 10, OUTPUT); // anodes
  pinMode(row_col[i] % 10, OUTPUT); // cathodes
}
pinMode(13, OUTPUT);
pinMode(12, OUTPUT);
pinMode(SwPower, OUTPUT);
digitalWrite(SwPower, HIGH);
beep(200);
for (int i = 0; i < buttNum; i++) pinMode(buttonArr[i], INPUT_PULLUP);

cli(); //Выкл всех прерываний
//set timer0 interrupt
TCCR0A = 0; // set entire TCCR2A register to 0
TCCR0B = 0; // same for TCCR2B
TCNT0 = 0; //initialize counter value to 0
// set compare match register for 2khz increments
OCR0A = 254; // = (16*10^6) / (2000*64) - 1 (must be <256)
TCCR0A |= (1 << WGM01); // turn on CTC mode
TCCR0B |= (1 << CS01) | (1 << CS00); // Set CS01 and CS00 bits for 64 prescaler
TIMSK0 |= (1 << OCIE0A); // enable timer compare interrupt
//set timer1 interrupt at 1Hz
TCCR1A = 0; // set entire TCCR1A register to 0
TCCR1B = 0; // same for TCCR1B
TCNT1 = 0; //initialize counter value to 0
// set compare match register for 1hz increments
OCR1A = 7812; // = (16*10^6) / (1*1024) - 1 (must be <65536)
TCCR1B |= (1 << WGM12); // turn on CTC mode
TCCR1B |= (1 << CS12) | (1 << CS10); // Set CS12 and CS10 bits for 1024 prescaler
TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
sei(); //Вкл всех прерываний
}

ISR(TIMER0_COMPA_vect) { //функция прерывания timer0
cli();
static int count;
for (int i = 0; i < 9; i++) //all Off
{
  digitalWrite(row_col[i] / 10, LOW); //row
  digitalWrite(row_col[i] % 10, HIGH); //col
}

if (count > 9 || count < 0 ) count = 0;
digitalWrite(row_col[count] / 10, HIGH);
if (data_C[count] ) digitalWrite(row_col[count] % 10, LOW);
count++;
sei();
}
}

```



```

delay_send++;
ReadKey();
beep(0);
}

ISR(TIMER1_COMPA_vect) {
    if (toggle_LED13) { //функция прерывания функция прерывания timer1 1Hz
        digitalWrite(13, HIGH); //ВКЛ pin 13 (LED)
        data_C[8] = toggle_LED13; // led 9
        toggle_LED13 = 0;
    }
    else { //ВыКЛ pin 13 (LED)
        digitalWrite(13, LOW);
        data_C[8] = toggle_LED13; //led 9
        toggle_LED13 = 1;
    }
    if (data_C[9] == 1 && data_C[10] == 1) data_C[8] = 1;
    if (data_C[10] == 0) {
        data_C[8] = 0;
        data_C[9] = 0;
    }

    if (delayStart > 0 && toggle_LED13) delayStart--; // декремент задержки если она больше 0
    if (data_C[9] == 1 && delayStart > 0) data_C[4] = !data_C[4]; // мигаем, показываем что вкл отсрочка
запуска
    if (data_C[9] == 1 && delayStart < 1) data_C[4] = 0; // гасим индикатор задержки и инд. времени задержки
}
//=====================================================================
void loop()
{
    updateData();
    sendData();
    checkErr();

}
//=====================================================================
void ReadKey()// Функция чтения состояния кнопок
{
    static byte drebezg8, drebezg9, drebezg10, drebezg11, drebezg15, drebezg16;
    byte countButtonPres = 0;
    /*
    if (!digitalRead(15)) drebezg15++; // Кнопка "старт"/"стоп"
    if (drebezg15 && digitalRead(15) )
    {
        drebezg15 = 0;
        data_C[9] = !data_C[9]; //start reset button data_C[9]
        if (!data_C[9])
        {
            for (uint8_t i = 0; i < 9; i++)
            { // сброс всех настроек и состояний
                data_C[i] = 0;
                Command_Massive[i] = 0;
            }
        }
        countButtonPres++;
    }
    /*
    if (!digitalRead(16)) drebezg16++; // Кнопка "питание" data_C[10]
    if (drebezg16 && digitalRead(16) )
    {
        drebezg16 = 0;
        data_C[10] = !data_C[10];
        if (!data_C[10])
        {
            for (uint8_t i = 0; i < 9; i++)
            { // сброс всех настроек и состояний
                data_C[i] = 0;
                Command_Massive[i] = 0;
            }
        }
        countButtonPres++;
    }
    if (countButtonPres > 0) beep(keyBeepTime);
    /*
    if (data_C[runRes] == 1 || data_C[power] == 0) return; // выход если не нажата кнопка
    /*
    if (!digitalRead(11)) drebezg11++; // Кнопка таймера отсрочки начала стирки
    if (drebezg11 && digitalRead(11))

```



```

{
    drebezg11 = 0;
    data_C[4]++;
    if (Command_Massive[Start] == ON) data_C[4] = Command_Massive[Status];
    switch (data_C[4])
    {
        case 0:
            delayStart = 0;
            break;
        case 1:
            delayStart = 120; //7200;
            break;
        case 2:
            delayStart = 14400;
            break;
        case 3:
            delayStart = 21600;
            break;
        case 4:
            delayStart = 32400;
            break;
        case 5:
            delayStart = 32400; // ))
            break;
        default:
            data_C[4] = delayStart = 0;
    }
    countButtonPres++;
}
/*
if (!digitalRead(9)) drebezg9++; // Кнопка "быстрой" стирки
if (drebezg9 && digitalRead(9) )
{
    drebezg9 = 0;
    data_C[6] = !data_C[6];
    if (data_C[5]) data_C[6] = 0;
    countButtonPres++;
}
/*
if (!digitalRead(10)) drebezg10++; // Кнопка "супер" стирки
if (drebezg10 && digitalRead(10) )
{
    drebezg10 = 0;
    data_C[5] = !data_C[5];
    if (data_C[6]) data_C[5] = 0;
    countButtonPres++;
}
/*
if (!digitalRead(8)) drebezg8++; // Кнопка двойное полоскание
if (drebezg8 && digitalRead(8) )
{
    drebezg8 = 0;
    data_C[7] = !data_C[7];
    countButtonPres++;
}
/*
if (countButtonPres > 0) beep(keyBeepTime); // Звук нажатия кнопок
}
//=====================================================================
int get_selector_position(int selector)//ФУНКЦИЯ чтения положения селекторов температуры, скорости
{
    int value = analogRead(selector);
#ifdef DEBUG
    softSerial.print("selector "); softSerial.println(selector);
    softSerial.print("U "); softSerial.println(value);
#endif
    if (selector == selectorProg)
    {
        for (int i = 0; i < progNum; i++)
        {
            if (value < selectorProg_values[0][i])
                return i;
        }
    }

    for (int i = 0; i < tempSpeedNum; i++)
    {
        if (value < selectorTempSpeed_values[0][i])
            return i;
    }
}

```



```

}

    return -1;
}
//=====================================================================
void serialEvent() // функция прерывания получения символа в UART
{
    if (Serial.available() > (byte)sizeof(Command_Massive)) // Если пришло сообщение размером с нашу
структуру
    {
        byte tmp[(byte)sizeof(Command_Massive)] = {0,};
        Serial.readBytes((byte*)&tmp, (byte)sizeof(Command_Massive));

        byte crc = 0;
        for (byte i = 0; i < 20 - 1; i++) crc ^= tmp[i];
        crc ^= 20 + 2;
#if(DEBUG)
        softSerial.print("\ncrc ");
        softSerial.print(crc);
#endif
        if (tmp[(byte)sizeof(Command_Massive) - 1] != crc || tmp[0] != ControlPanelID)
        {
            errConn++;
#if(DEBUG)
            softSerial.print(" ERR ");
            softSerial.print(errConn);
#endif
            return;
        }
        memcpy((byte*)&Command_Massive, tmp, sizeof(tmp));
#if(DEBUG)
        softSerial.print("\n>> ");
        for (byte i = 0; i < sizeof(Command_Massive); i++)
        {softSerial.print(Command_Massive[i]); softSerial.print(" ");}
#endif
#endif
        flag_send = true; // Поднимаем флаг отправки данных
        delay_send = 0; // Сброс задержки перед отправкой данных
    }
}

//=====================================================================
void sendData() // Функция отправки данных мастеру
{
    if (flag_send == false || delay_send < 50) return; // Если флаг отправки не поднят
                                                       // или задержка отправки не прошла
                                                       // выходим из функции отправки данных

    byte tmp_buf[sizeof(Command_Massive)] = {0,};
    tmp_buf[ID] = MasterID;
    tmp_buf[Start] = Command_Massive[Start];//data_C[runRes];
    tmp_buf[Prog] = 10;//Command_Massive[Prog]
    tmp_buf[Temper] = 40;//Command_Massive[Temper]
    tmp_buf[Spin] = 50;//Command_Massive[Spin]
    for (byte i = 0; i < 20 - 1; i++) // Подсчет CRC отправляемых данных
        tmp_buf[19] ^= tmp_buf[i];
    tmp_buf[19] ^= 20 + 2;
#if (DEBUG)
    softSerial.print("\n<< ");
    for (byte i = 0; i < sizeof(tmp_buf); i++)
    {softSerial.print(tmp_buf[i]);softSerial.print(" ");}
#endif
    #endif
    Serial.write((byte*)&tmp_buf, sizeof(tmp_buf));// Отвечаем контроллеру (мастеру)
    flag_send = false; // опускаем флаг отправки данных мастеру
}
//=====================================================================
void updateData() // Функция обработки данных состояния компонентов управл.
{
    if (data_C[runRes] && delayStart < 1)
    {
        Command_Massive[Start] = ON;
    } else {
        Command_Massive[Start] = OFF;
    }
    if (Command_Massive[FlagEnd] == 1 && data_C[runRes] == 1 && delayStart < 1)
    {
        data_C[runRes] = 0;
        delayStart = 0;
        Command_Massive[FlagEnd] = 0;
        beep(500);
    }

    if (data_C[runRes] == 0 && data_C[power] == 1)

```



```

{
    Command_Massive[Prog]      = selectorProg_values [1] [get_selector_position(selectorProg)];
    Command_Massive[Temper]     = selectorTempSpeed_values [1] [get_selector_position(selectorTemp)];
    Command_Massive[Spin]       = selectorTempSpeed_values [2] [get_selector_position(selectorSpeed)];
}
byte tmp = 0;
if (Command_Massive[Start] == ON) {
    tmp = Command_Massive[Status];
} else {
    tmp = data_C[4];
}
// зажигаем индикаторы состояния стирки помещая в массив значение
switch (tmp)
{
    case 0:
        data_C[0] = 0; data_C[1] = 0; data_C[2] = 0; data_C[3] = 0;
        break;
    case 1:
        data_C[0] = 1; data_C[1] = 0; data_C[2] = 0; data_C[3] = 0;
        break;
    case 2:
        data_C[0] = 0; data_C[1] = 1; data_C[2] = 0; data_C[3] = 0;
        break;
    case 3:
        data_C[0] = 0; data_C[1] = 0; data_C[2] = 1; data_C[3] = 0;
        break;
    case 4:
        data_C[0] = 0; data_C[1] = 0; data_C[2] = 0; data_C[3] = 1;
        break;
    default:
        Command_Massive[Status] = 255;
}
}

/*=====
void beep(unsigned int hold)// Функция звуковой индикации
{
    longlastTime++;
    static unsigned int holdTime;
    if (hold > 0)
    {
        digitalWrite(Beep, HIGH);
        holdTime = hold;
        longlastTime = 0;
    }
    if (longlastTime >= holdTime)
    {
        holdTime = 0;
        digitalWrite(Beep, LOW);
    }
}

=====*/
void checkErr()// Функция инициализации послед. порта
{
    if (errConn < 1) return;
    errConn = 0;
    Serial.end();
    Serial.begin(9600);
}
=====*/

```

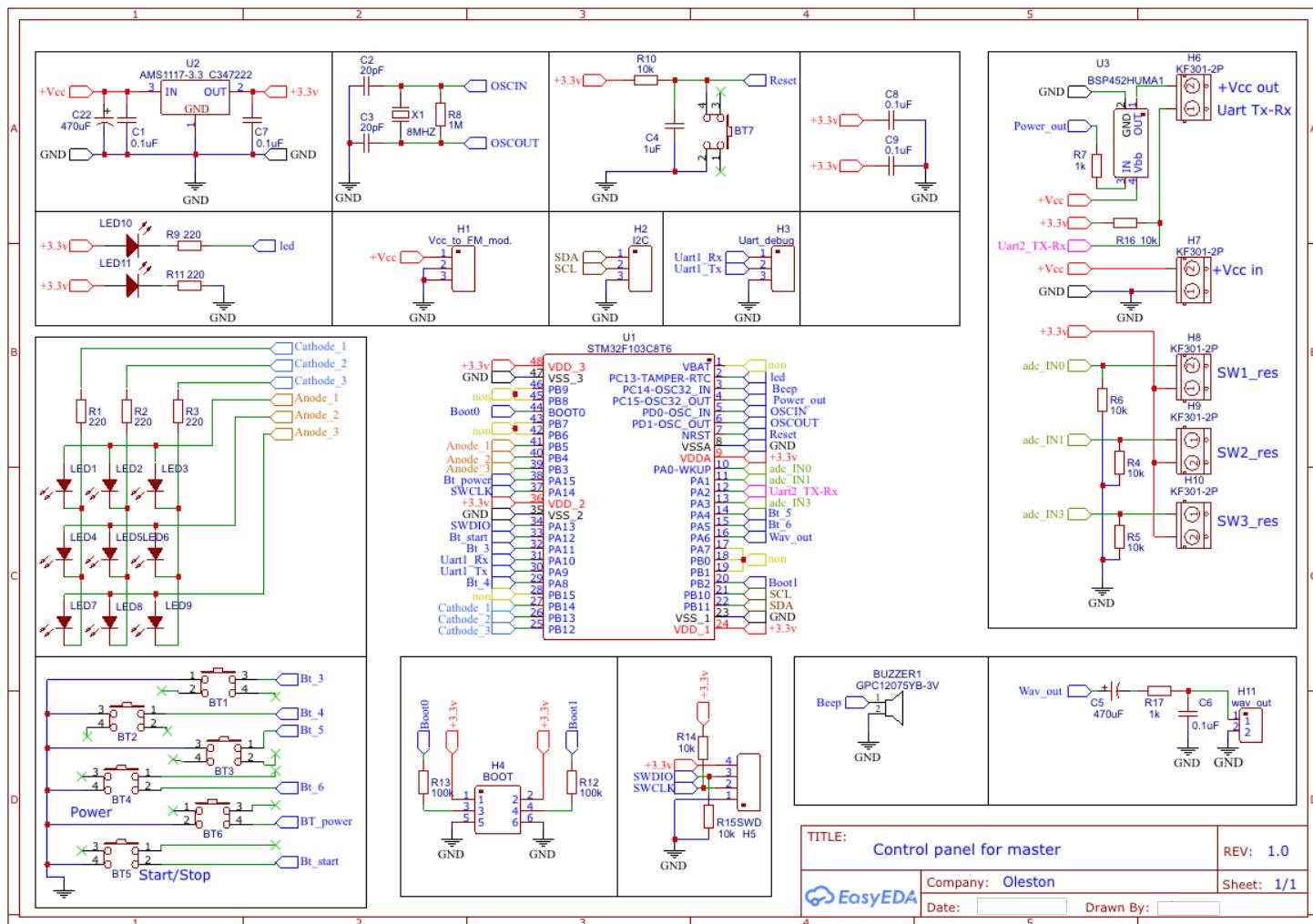
Вышеприведенный пример №2 имеет более высокий функционал по индикации и элементам управления СМ в отличии от реализации из примера № 1. Контроллер управления СМ, примера №2, повторяет функционал ранее установленного на СМ в заводском исполнении, за исключением ряда функций, например, как дополнительное полоскание и др. Подобными функциями легко дополнить логику контроллера управления имея четкое представление об алгоритме их.



Пример №3.

Управление контроллером CM (Ariston AVSL 100) по однопроводному интерфейсу на базе микроконтроллера STM32F103C8T6.

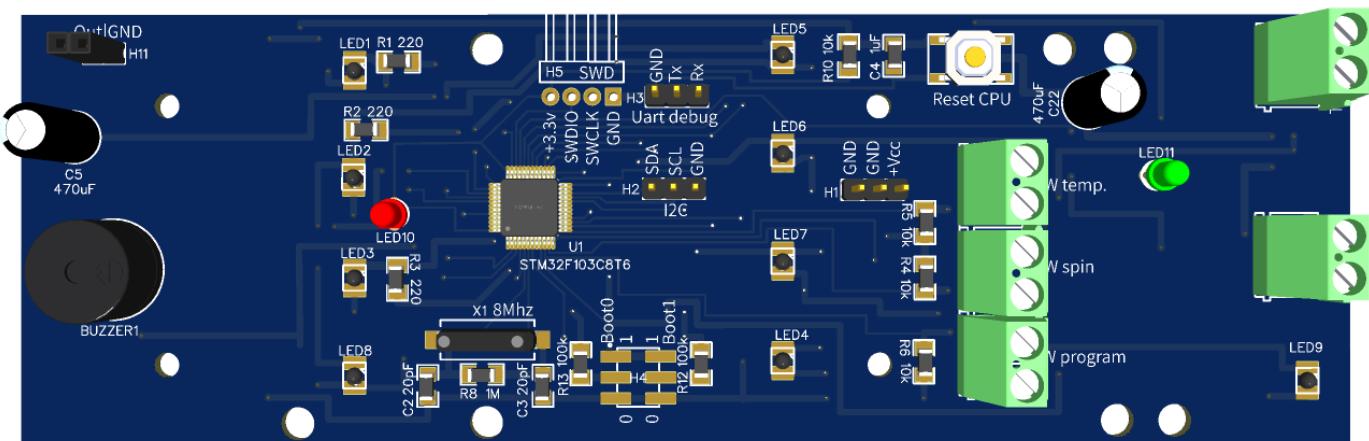
Принципиальная схема контроллера панели управления СМ (Ariston AVSL 100) на базе STM32F103C8T6.



Макет печатной платы контроллера панели управления СМ (Ariston AVSL 100) на базе контроллера STM32F103C8T6.*

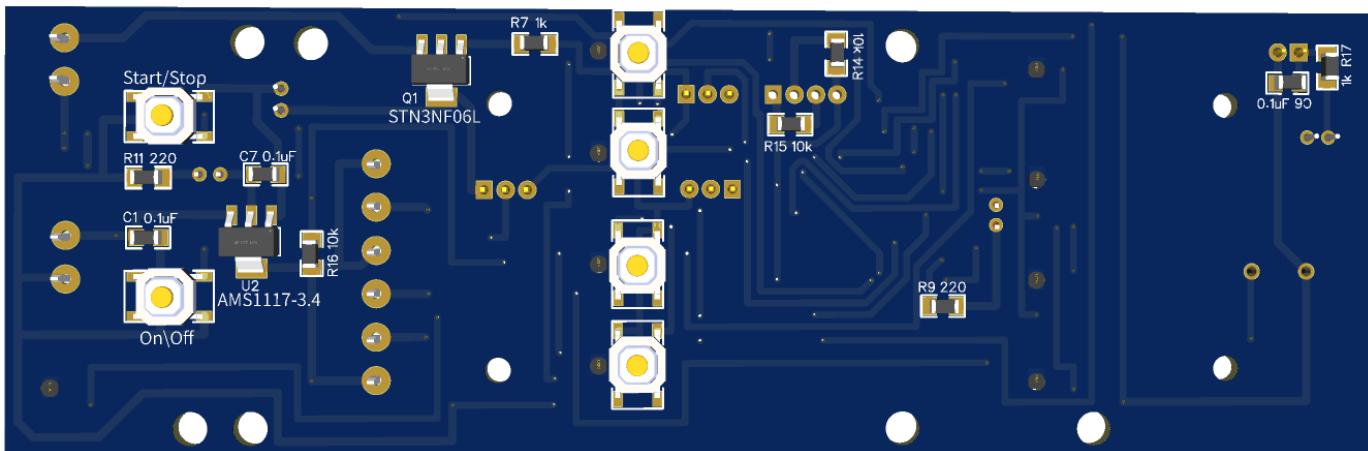
Размер печатной платы изготовлен под размер штампной платы контроллера управления СМ.

Верхняя сторона.





Макет печатной платы контроллера панели управления CM (Ariston AVSL 100) на базе контроллера STM32F103C8T6
Нижняя сторона.

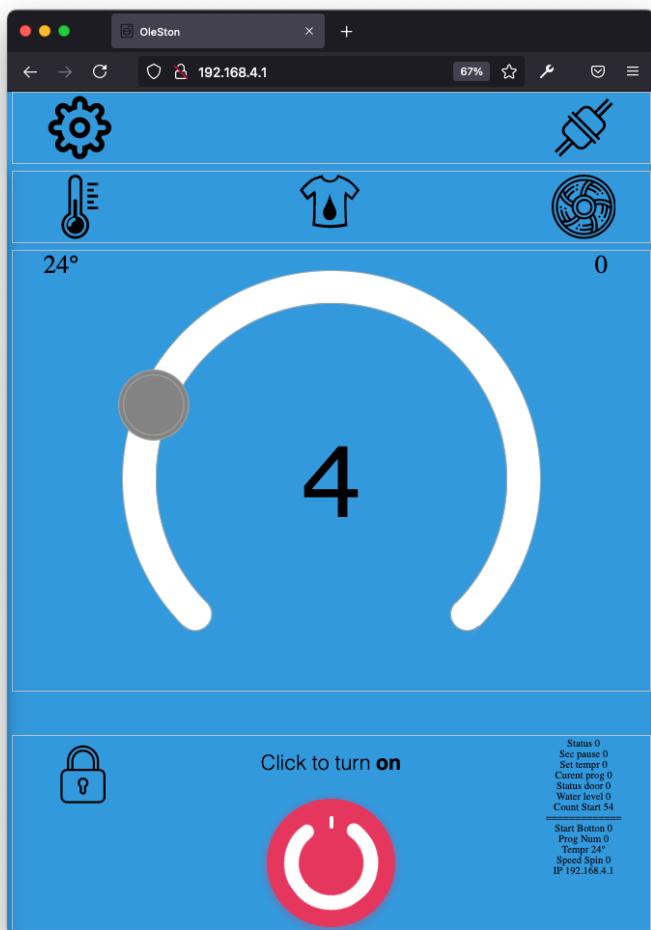


Реализация панели управления на микроконтроллере STM32F103C8T6 более технологична и имеет хороший потенциал по наращиванию функционала по управлению контроллером СМ, так и дополнительного, например, управлением модулем FM радио для прослушивания радиостанций или Bluetooth трансляций. Код панели, построенный в среде STM32CubeIDE, на языке Си, доступен для скачивания в разделе «Ссылки на исходные тексты программ». Пример №3. Реализован только основной, управляющий функционал.
**Дополнительные возможности в виде поддержки модулей FM, Bluetooth будут реализованы дополнительно.*



Пример №4.

Управление контроллером СМ (Ariston AVSL 100) с помощью модуля ESP8266 подключенного по однопроводному интерфейсу.



На рисунке слева, приведен пример интерактивной панели управления контроллером СМ. Панель управления доступна с помощью веб браузера и состоит из html и java script кода. Источником интерактивного кода и веб сервера выступает контроллер ESP8266. Обсуждаемая панель содержит полнофункциональные элементы управления и индикации, необходимые для контроля и управления контроллером СМ. Данный подход упрощает вопрос восстановления и/или модернизации СМ, так как не требует дополнительных усилий в создании органов управления в виде кнопок, регуляторов на корпусе СМ.

Панель управления на ESP8266 может выступать в качестве точки доступа с фиксированным IP адресом, также, имеет режим клиента для подключения к сети WIFI. Код панели, построенный в среде Arduino IDE, доступен для скачивания в разделе «Ссылки на исходные тексты программ» Пример №4.

Ссылки на исходные тексты программы.

Пример №1 ссылка: <https://github.com/OleSton/Simple-control-panel-of-OleSton-with-Arduino-Nano>

Пример №2 ссылка: <https://github.com/OleSton/Control-Panel-with-Arduino-Nano>

Пример №3 ссылка: <https://github.com/OleSton/OleSton-control-panel-with-STM32F103C8T6>

Пример №4 ссылка: <https://github.com/OleSton/ESP8266-Control-Panel-OleSton>

Пример №5 ссылка:



Регулировка, индикация, ошибки контроллера СМ.

ПИД регулятор двигателя уже имеет усреднённые настройки для большинства коллекторных двигателей. Настройки, коэффициенты хранятся в файле, расположенному по следующему пути "**MOT/PID**". Стока в файле: 45.0;1.5;0.0; P=45.0, I=1.5, D=0.0. При необходимости изменения коэффициентов воспользуйтесь текстовым редактором, например «Блокнот».

Тахогенератор выполнен в виде нескольких индуктивных обмоток, находящихся вокруг постоянного магнита расположенного на валу двигателя. Количество этих обмоток отвечает за кол-во импульсов при одном полном обороте вала двигателя с магнитом. При низкой скорости вращения вала, амплитуда выдаваемых импульсов очень мала и обнаружить их затруднительно. Для опознания параметров тахогенератора лучше обратиться к документации на данное изделие. Информация о кол-ве выдаваемых импульсов от тахогенератора находится в файле по следующему пути "**MOT/TAHO**". Стока в файле: 8; Т.е. 8 импульсов на один оборот.

Редукция. Количество оборотов вала двигателя к одному обороту барабана СМ хранится в файле по следующему адресу "**MOT/REDU**". Стока в файле: 14;. Для измерения кол-ва оборотов вала двигателя сделайте на нем метку, для лучшего определения, медленно проверните на один полный оборот барабан, в процессе поворота подсчитайте сколько полных оборотов сделал вал двигателя. Если значение отлично от занесенного в файл, то с помощью текстового редактора измените на ваше полученное.

На плате контроллера присутствуют два световых индикатора.

Первый индикатор находится на понижающем блоке питания контроллера и индицирует о наличии напряжения питания.

Второй индикатор имеет маркировку "LED1", расположен в нижней правой части печатной платы контроллера СМ. При нормальной работе "LED1" меняет свое значения один раз в секунду, т.е. мигает. При частом мигании, чаще чем один раз в секунду, сигнализирует об отсутствии SD карты в карта приемнике или неподдерживаемом формате карты. Постоянно горение или отсутствие его говорит о том, что контроллер в неисправном состоянии.

При обнаружении известных ошибок происходил их фиксация в файле LOG.LOG.

Лог файл (LOG.LOG) может содержать следующие ошибки:

Ошибки контроллера СМ.

ID Ошибки	Регистрируемая строка в LOG.LOG файле	Описание ошибки
200	"problem of filling the tank with water"	Проблема заполнение водой бака до уровня 1. Если уровень 1 не достиг за 360 секунд, при команде подачи воды, то возникает ошибка с остановкой процесса стирки. Для панели управления выставляется статус 200.
201	"water drain problem"	Проблема со сливом воды из бака. Возникает если уровень воды не становится меньше 1 по истечении 360 секунд. Процесс стирки прерывается. Для панели управления выставляется статус 201.
202	"water heating problem"	Проблема с нагревом воды. Если заданная температура не достигает за 20 мин своего значения, то возникает ошибка. Процесс стирки прерывается. Для панели управления выставляется статус 202.
203	"temperature sensor problem"	Ошибка возникает при неисправности датчика температуры или линии связи с ним. Процесс стирки прерывается. Для панели управления выставляется статус 203.
204	"high water temperature"	Ошибка возникает при выходе показаний температуры за пределы: меньше 3°C и больше



		104 °C. Процесс стирки прерывается. Для панели управления выставляется статус 204.
205	"motor rotation error"	Проблема с вращением ротора мотора – мотор не вращается. Процесс стирки прерывается. Для панели управления выставляется статус 205.
	"E1"	Ошибка файловой системы.
	"E2"	Ошибка записи данных в память.
	"E3"	Ошибка записанных данных в память.
	"E4"	Ошибка целостности данных в памяти.

Контроллер СМ имеет возможность воспроизводить звуковые файлы формата WAV, расположенные в одноименной директории SD карты. Линейный выход аудио сигнала представлен на печатной плате контроллера разъемом H11. Выходной сигнал с разъёма H11 слаб для подключения к нему звукового излучателя. Для получения соответствующей мощности звукового сигнала необходимо использовать усилитель (УМЗЧ). Команда позволяющая проиграть звуковой файл выглядит следующим образом **T5**: «T» - команда, 5 – имя файла (WAV/5.WAV). Команда «блокирующая», т.е. пока файл не будет проигран до конца выполнение следующей команды не будет выполняться.

Перечень элементов и номиналов для самостоятельного монтажа.

Обозначенная ориентировочная стоимость компонентов, в последних двух столбцах, указана в долларах США для оценки себестоимости изделия.

Перечень электронных компонентов СМ.

ID	Designator	Name	Footprint	Quantity	Price	
1	220V,TAHO	KF301-2P	KF301-5.0 2P	2	0,08	0,16
2	ANALOG_IN,HEATER,PUMP,TO_PANEL_C ONTROL1,VALVE1,VALVE2,VALVE3	KF301-2P	CNT-KF301-2P (3MM PAD) COPY	7	0,08	0,56
3	BP1,R38,R39	0	1206	3	0,01	0,02
4	BUZZER1	GPC12075YB-3V	BEEP-DIP-12MMX7.5MM	1	0,29	0,29
5	C1,C2,C3,C5,C6,C8,C20,C22,C23,C26,C30	0.1uF	1206	11	0,02	0,18
6	C11,C12,C13	10uF	2312	3	0,01	0,03
7	C14	1uF	C1206	1	0,00	0,00
8	C15,C16,C17,C18	20pF	C1206	4	0,00	0,02
9	C19	100uF	CASE-C_6032	1	0,03	0,03
10	C21	330uF	CASE-C_6032	1	0,03	0,03
11	C25	470uF	2312	1	0,03	0,03
12	C27,C28,C29	0.047uF	CAP-HV-9.0*3.0	3	0,03	0,10
13	C4,C10	220nF	CAP-18*5-15MM	2	0,08	0,16
14	C7,C24	470uF	CAP-D6.3XF2.5	2	0,03	0,05
15	D1	1N4148	SMAJ-5.5X2.6	1	0,01	0,01
16	D2	STPS1L30A	SMA_L4.3-W2.6-LS5.2-RD	1	0,52	0,52
17	DOORLOCK,POWER,PRESOSTAT,TEMP_D S	JR KF301-3P	CNT-KF301-3P	4	0,08	0,32
18	F1	Fuse seat	5MM*20MM	1	0,13	0,13
19	H11	wav_out	HDR-TH_2P-P2.54-V-M	1	0,07	0,07

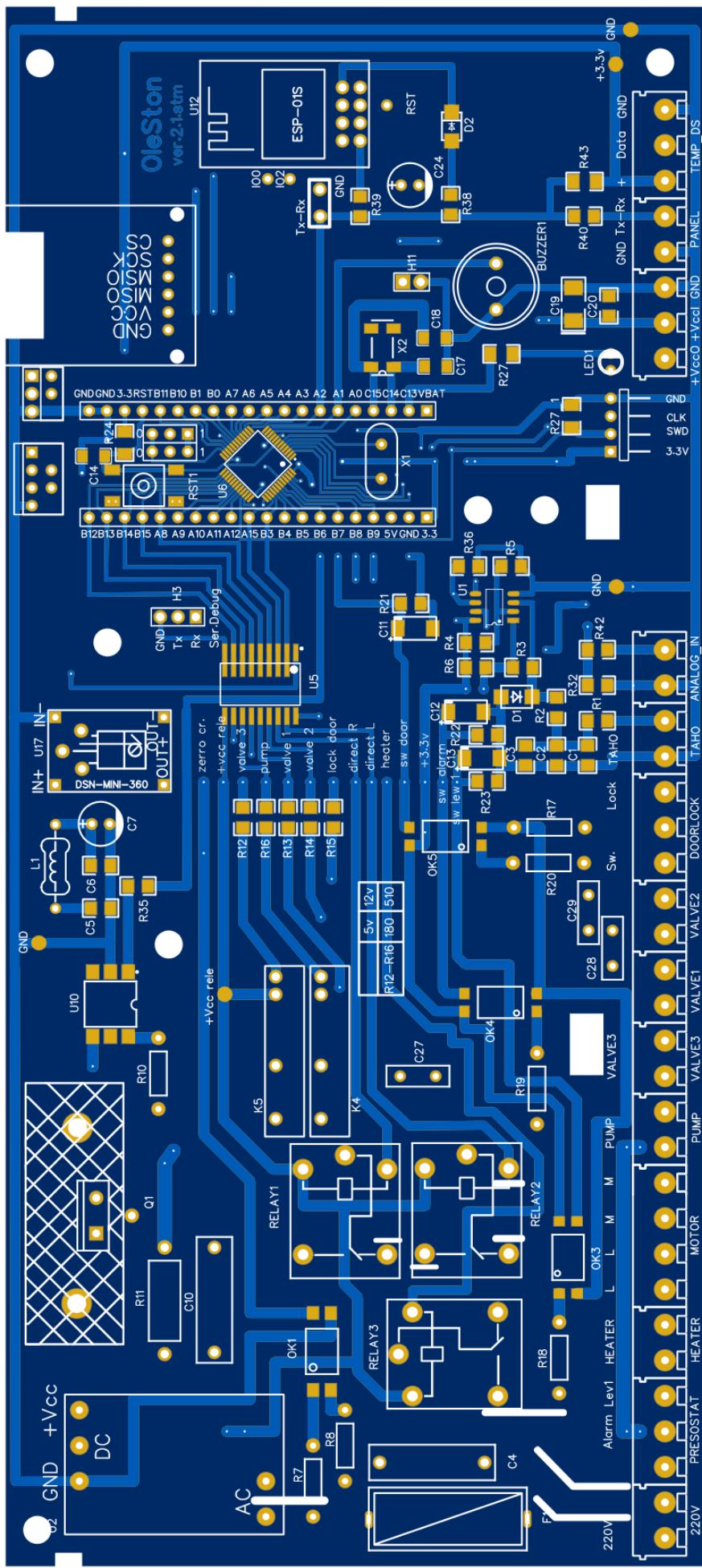


20	H3	Uart_debug	HDR-TH_3P-P2.54-V	1	0,05	0,05
21	IO0,IO2,RST	io0	1PIN	3	0,00	0,00
22	K4,K5	G3MB-202PL	G3MB-202P	2	0,75	1,50
23	L1	L-USCECL	L_CECL	1	0,05	0,05
24	LED1	3AR2SC10 90-120	LED-3MM	1	0,01	0,01
25	MOTOR	KF301-3P	CNT-KF301-4P	1	0,08	0,08
26	OK1	PC814	SMD-4(6.5X4.58)	1	0,09	0,09
27	OK3,OK4,OK5	PC817	SOP-4(6.5X4.58)	3	0,09	0,27
28	P1	HDR-IDC-2.54-2X3P	HEADER-2.54-P2x3	1	0,07	0,07
29	P2	SWD	HEADER-2.54-P4-ANG	1	0,02	0,02
30	P3,P4	210S-1*20P L=11.6MM Gold-plated black	HEADER-2.54-P20	2	0,10	0,21
31	Q1	BTA16-800BWRG	TO-220 КОНТАКТЫ PA3HECEНЫ	1	0,83	0,83
32	R1	3k3	1206	1	0,01	0,01
33	R10	380	AXIAL-0.4	1	0,01	0,01
34	R11	47	AXIAL-0.6	1	0,01	0,01
35	R12,R13,R14,R15,R16	360*	1206	5	0,01	0,03
36	R17	10k	AXIAL-0.4	1	0,06	0,06
37	R18,R19,R20	100k	AXIAL-0.4	3	0,06	0,18
38	R2	2k2	1206	1	0,01	0,01
39	R24,R27_1,R30	10k	1206	3	0,01	0,02
40	R25,R26	100k	1206	2	0,01	0,01
41	R27	510	1206	1	0,00	0,00
42	R28	1M	1206	1	0,01	0,01
43	R29	100k*	1206	1	0,01	0,01
44	R3	100k	1206	1	0,01	0,01
45	R31,R32,R41	4k7*	1206	3	0,01	0,02
46	R34	1k	1206	1	0,01	0,01
47	R35	360	1206	1	0,01	0,01
48	R36,R37,R42	10k*	1206	3	0,01	0,02
49	R4,R21,R22,R23,R33	4k7	1206	5	0,01	0,03
50	R40	100	1206	1	0,01	0,01
51	R5	470	1206	1	0,01	0,01
52	R6,R9,R43	10k	1206	3	0,01	0,02
53	R7,R8	31K	AXIAL-0.4	2	0,01	0,01
54	RELAY1,RELAY2,RELAY3	SRD-05VDC-SL-C	RELAY-SL-SRD	3	0,36	1,08
55	RST1	K2-6639SP-C4SC-04	KEY-6.0*6.0	1	0,04	0,04
56	U1	LM393	SOP8	1	0,04	0,04
57	U10	MOC3023SM	SMD-6_L7.3-W6.5-P2.54-LS10.16-BL	1	0,37	0,37
58	U11	Micro SD Card Adapter	MICRO SD CARD ADAPTER	1	0,50	0,50
59	U12	ESP-01S_C82893	ESP-01S	1	1,10	1,10
60	U13,U14,U15	MOC3062S-TA1	OPTO-SMD-6_L8.5-W6.5-P2.54-LS9.2-BL	3	0,19	0,58

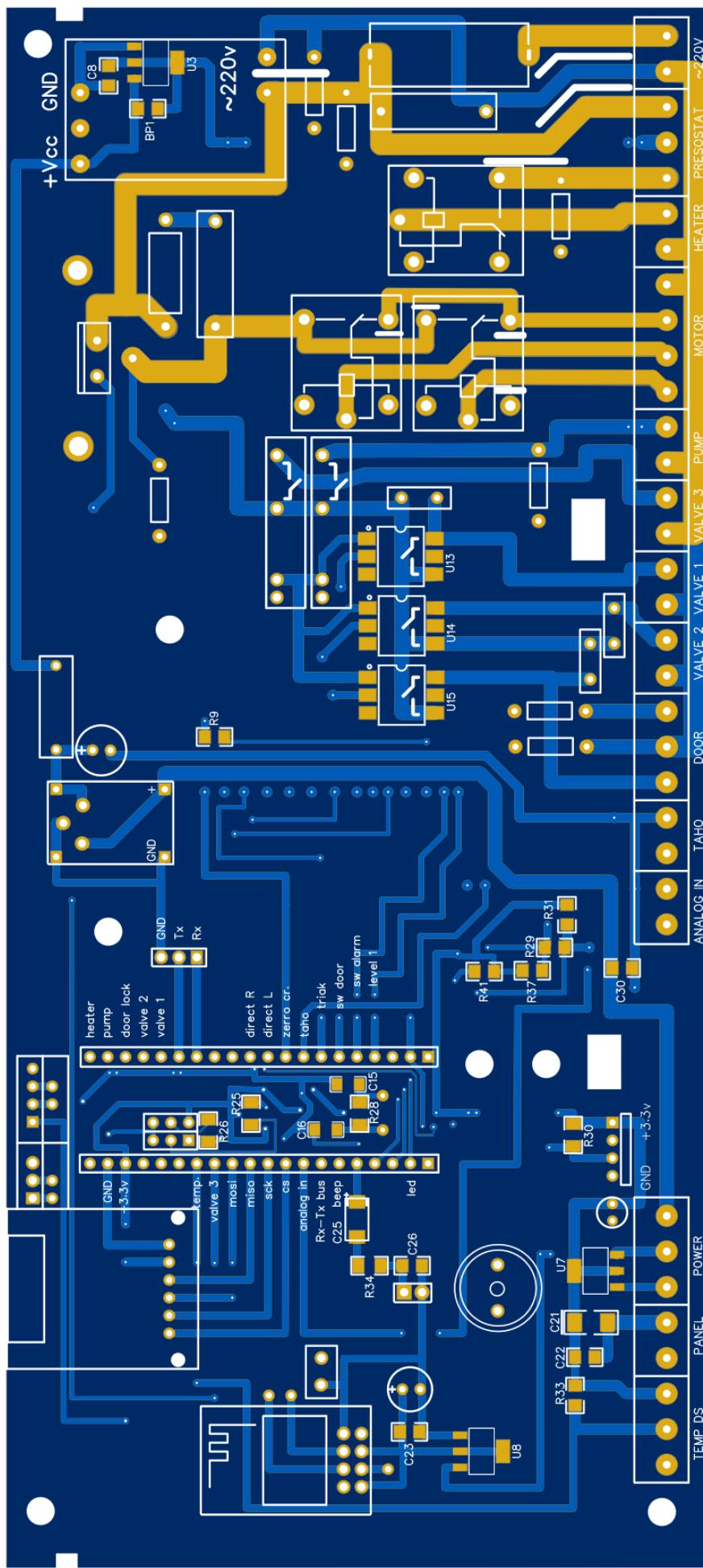


61	U17	MINI-360	DSN-MINI-360	1	0,40	0,40
62	U2	Chinese AC/DC converter	CHINESE AD/DC 5V 700MA	1	1,25	1,25
63	U3,U7,U8	AMS1117-3.3	SOT-223	3	0,10	0,30
64	U5	ULN2803AD	SOP-18_L11.4-W7.6-P1.27-LS10.6-BL	1	0,15	0,15
65	U6	STM32F103C8T6	ST-LQFP48	1	7,00	7,00
66	X1	8MHz	OSC-49S-1	1	0,07	0,07
67	X2	32.768KHz	MC-306	1	0,31	0,31
						Total price \$
						19,53

Вид печатной платы контроллера СМ.



Верхняя сторона печатной платы. (ver.2.1)



Нижняя сторона печатной платы. (ver.2.1)



Программирование контроллера управляющей программой.

Программа микроконтроллера состоит из двух частей:

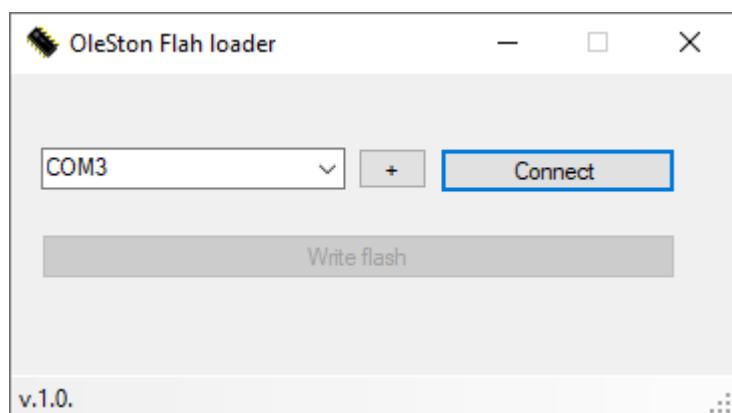
Первая – программа загрузчик, которая позволяет осуществлять загрузку основной управляющей программы контроллера, также обновлять основную управляющую программу через SD – карту.

Вторая – собственно основная программа управления контроллера СМ.

Для загрузки программного обеспечения в новый («чистый») контроллер необходимо сначала загрузить загрузчик (BOOT LOADER).

Программирование загрузчика осуществляется с помощью программного обеспечения **OleSton Flash loader** и последовательного порта с конвертором логических уровней напряжением 3.3 вольта, например типа USB-TTL-PL2303HX.

Интерфейс и последовательность действий программы **OleSton Flash loader** при загрузке загрузчика в память микроконтроллера СМ.



Подключить конвертор USB-UART к порту Н3 платы контроллера под названием «Ser.Debug».

Подключить конвертор USB-UART к ПК с которого будет осуществляться программирование контроллера СМ.

Убедиться, что ОС определила конвертор USB-UART и готова к взаимодействию с ним.

Запустить программу **OleSton Flash loader**.

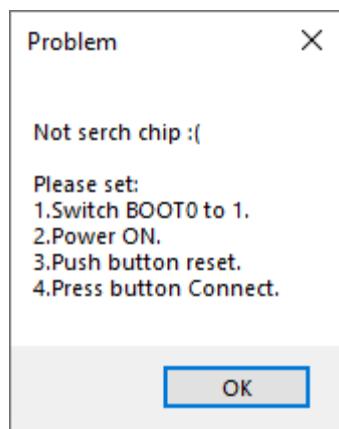
Выбрать последовательный порт, соответствующий конвертору USB-UART.

Установить перемычки **BOOT** в положение **BOOT0=1, BOOT1=0**.

Подать питающее напряжение на контроллер СМ.

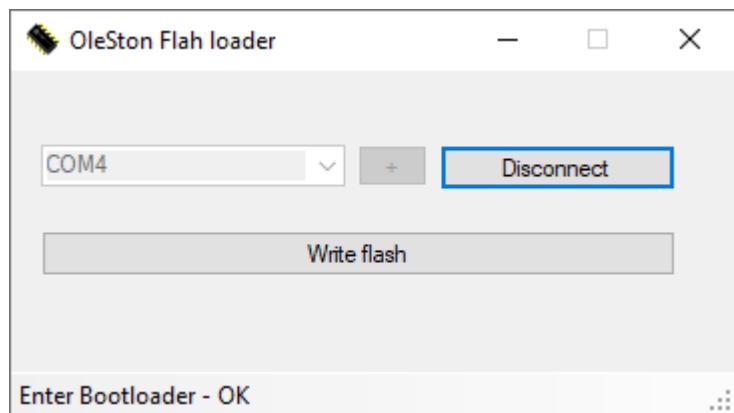
Нажать кнопку **RESET** на 2 секунды.

На ПО **OleSton Flash loader** нажать кнопку **“Connect”**.

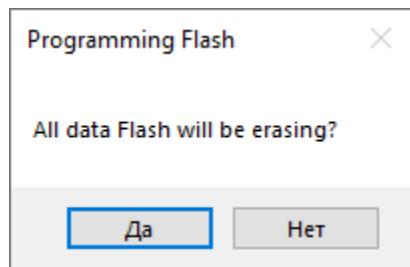


При появлении окна как на рисунке слева, говорит о возникших проблемах с соединением или вводом контроллера в режим программирования.

Возможные причины — это неправильное соединение линий связи между конвертором USB-UART и «Ser.Debug». Отсутствие питающего напряжения микроконтроллера. Неправильно выбран последовательный интерфейс. Не проведенная процедура ввода микроконтроллера в режим программирования. Неисправность контроллера или обвязывающих его элементов.

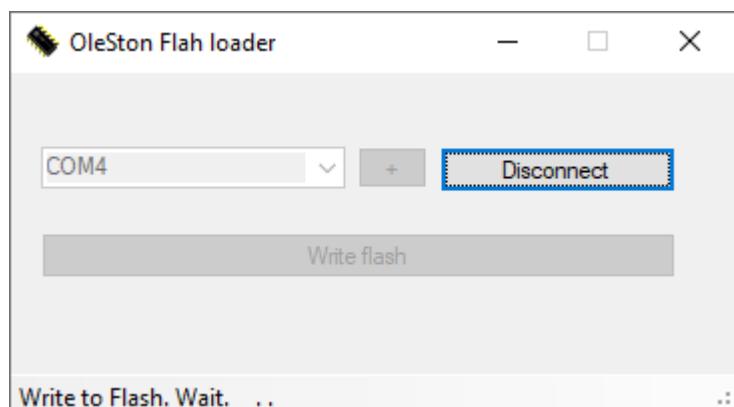


Признаком удачного соединения ПО **OleSton Flash loader** с контроллером будет свидетельствовать надпись (Enter Botloader - OK), в нижней части интерфейса ПО. Надпись сообщает о том, что выполнен вход в режим программирования контроллера СМ. Для начала процесса программирования контроллера СМ необходимо нажать кнопку "Write flash".

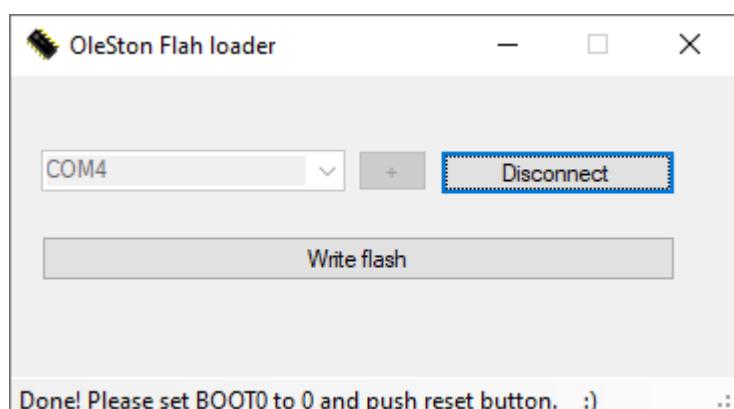


После нажатия кнопки "Write flash" требуется подтверждение своих действий в программировании контроллера СМ. Сообщение на картинке слева запрашивает подтверждение на очистку памяти контроллера для последующего программирования его.

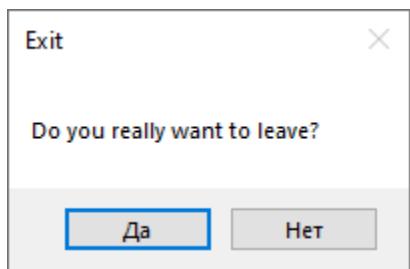
Внимание!!! Микроконтроллер будет полностью очищен от предыдущего программного обеспечения и восстановление последнего будет невозможно. Если Вы ошибочно нажали кнопку "Write flash", то есть возможность отменить стирание и программирование на данном этапе.



Процесс программирования контроллера СМ занимает примерно 1-2 минуты. Во время программирования не допускается разъединение соединительных линий между контроллером СМ и ПК, равно как отключение питающего напряжения на обоих устройствах. В процессе программирования, в нижней части ПО **OleSton Flash loader** будет присутствовать надпись с периодически появляющимися точками, которые свидетельствуют, что процесс находится в работе.



После завершения программирования в нижней части ПО **OleSton Flash loader** появится надпись о завершении процедуры и о необходимости вернуть перемычку **BOOT0** в положение **0** с последующей нажатием кнопки **Reset** для перезагрузки контроллера СМ.



Слева окно появляющееся при закрытии ПО **OleSton Flash loader**. Для выхода и ПО необходимо нажать кнопку с положительно утвердительным ответом. В противном случае нажать другую кнопку.

Загрузка основной программы управления контроллера СМ осуществляется с помощью размещения файла программы на SD – карте, в директории **FRMW**. Имя файла программы должно быть следующим: **PROG.BIN**. При обнаружении в папке **FRMW** файл с именем **PROG.BIN** во время включения контроллера, начинается процесс обновления, с последующим переименованием файла на **PROG.BI_**

Документ в стадии редактирования...
30/07/21