## Предлагаемая модель протокола для общения контроллера и сервера

Все запросы инициирует контроллер, например каждые 10 сек + при изменении любого из его параметров (включение входа, выхода, изменение температуры и т.д.).

Запросы могут быть отправлены get/postзапросом, адрес скрипта будет http://controller.alpachini.com/api.php?request=наш\_пакет\_от \_контроллера\_в\_json\_и\_зашифрованный\_и\_переведенный\_в\_hex\_формат

По данному адресу скрипт появится в ближайшее время (1-2 дня) и, временно, можно будет в него посылать пакет как шифрованный, так и не шифрованный, просто json в hexформате, ответ в этом случае будет как шифрованный, так и не шифрованный – это для целей тестирования.

Шифрованный ответ будет приходить в hex формате, нешифрованный – чистым json текстом

1. Структура пакета, отправляемая контроллером на сервер (значение параметра request – это json следующего вида:

{

“id”:”AFHDCBSDSJKHSA8934743”, - Любая цифро-буквенная комбинация, уникальный ид контроллера,

“state”: {

“inputs”: [0, 0, 1,1,0,1,0,1…..], -цифровые входы, массив значений, 0-1

“analog\_inputs”: [251,25,125,0,…..], - аналоговые входы, массив значений

“relays”:[0, 1, 1…..], - цифровые выходы, реле, массив значений, 0-1

“analog\_outputs”: “[14,152,0,255…]”,– аналоговые выходы, текущее состояние, массив значений

“temp\_sensors”: [24,255,145,12….]– температурные сенсоры, массив значений

} ,

“command\_result”:””, - результат выполнения команды, пришедшей в предыдущем пакете – если она была, если не было команды – просто пустое значение

“firmware\_id”:”100”, - ид прошивки, числовое значение

“time”:1566682245 – текущее время контроллера, unixtimestamp

}

Возможные ответы контроллера:

Вслучае, если сервер или не понимает json, или отсутствуют какие-либо из полей в пакете запроса:

{

“result”:”error”, - метка result, будет в каждом ответе

“error\_text”:”Описаниеошибки”,– текст ошибки, больше для разработчиков, чтобы понимать, что не так

“time”:1566682245

}

Если запрос принят, понят и все хорошо, и команд контроллеру нет, то ответный пакет будет:

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:” ”, – команда контроллеру на выполнение

“time”:1566682245

}

Особенность поля time – синхронизация происходит автоматически, т.е. получив пакет с ответом и полем time и если значение этого поля больше, чем значение текущего времени контроллера–контроллер значение этого поля принимает за текущее время, свое внутреннее время устанавливает в это значение при каждом получении пакета от сервера.

Если значение этого поля меньше, чем текущее значение времени в контроллере – значит пакет пришел из прошлого, имеет место быть подмена пакета, точнее его повтор, и нужно его игнорировать

Если есть какая-то команда контроллеру от сервера:

## Установка значений цифровых выходов (реле)

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”relays”,

“value”:”[0,0,1,1,0,0,1….]”

Это команда установки значений цифровых выходов, включения реле, в состояние, указанное в массиве, т.е. первый выход -0 , второй выход 0, третий выход 1 и т.д.

}”,

“time”:1566682245

}

## Установка значений аналоговых выходов

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”analog\_ouputs”,

“value”:”[14,145,10,1,0,0,1….]”

Это команда установки значений аналоговых выходов, в состояние, указанное в массиве, т.е. первый выход -14 , второй выход 145, третий выход 10 и т.д.

}”,

“time”:1566682245

}

## Установка значений условий срабатывания выходов (по сути программирование контроллера)

Пока самый незавершенный и непроработанный раздел данного протокола, но модель предлагаю такую:

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”set\_program”,

“value”:”ACD4C4B2D2F0012FFAC….”, - байт-код в шестнадцатеричном представлении

“id\_program”:”2” - ид данной программы

}”,

“time”:1566682245

}

В чем заключается идея…. К сожалению, не получается судя по всему упростить нам жизнь, и вешать по одной программе на каждый вход или выход, не получается привязывать программу ко входу…. У нас выходит, что входных параметров (значений входов или выходов или температурных датчиков) может быть несколько…и выходных параметров может быть несколько, например может быть программа типа «если вход 1 = 0, вход 2=1 и аналоговый вход 5 = 120 – включить выход 1, выключить выход 3» - в таких условиях не привяжешь программу к конкретному входу или выходу никак…

Соответственно, возможно! придется иметь какое-то хранилище программ в контроллере, т.е. добавлять туда программы, удалять оттуда программы или заменять их... а чтобы их как-то различать, есть идея присваивать каждой программе какое-то ид, кроме собственно байт-кода.

Проще говоря, если мы добавляем программу с ид 2 – то она добавляется в хранилище контроллера с ид 2 и ее байт-кодом. Если программа с таким ид уже там существует – то она будет перезаписана – изменена.

Если мы ходим удалить эту программу из контроллера:

## Удаление программы из контроллера

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”set\_program”,

“value”:” ”, - пустое значение байт-кода

“id\_program”:”2” - ид данной программы

}”,

“time”:1566682245

}

Соответственно, сервер должен иметь возможность прежде чем позволять программировать контроллер – считывать список уже существующих в нем программ:

## Получение списка программ

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”list\_programs”

}”,

“time”:1566682245

}

В ответ контроллер в своем следующем пакете должен в поле command\_result поместить такой пакет:

…………………

“temp\_sensors”: [24,255,145,12….]– температурные сенсоры, массив значений

} ,

“command\_result”:

”{

“type”:”list\_programs”, - повторяется тип команды, на которую мы отвечаем

“value”:”[“14”,”10”,”16”]”– значение ответа, список ид сохраненных программ

}”,

“firmware\_id”:”100”,

…………………………….

## Запрос байт-кода конкретной программы

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”get\_program”,

“id\_program”:”2” - ид данной программы

}”,

“time”:1566682245

}

В ответ контроллер в своем следующем пакете должен в поле command\_result поместить такой пакет:

…………………

“temp\_sensors”: [24,255,145,12….]– температурные сенсоры, массив значений

} ,

“command\_result”:

”{

“type”:”get\_program ”, - повторяется тип команды, на которую мы отвечаем

“value”:”A1FD0C0E111F2514…”– значение ответабайт-кода

}”,

“firmware\_id”:”100”,

…………………………….

## Байт-код и собственно программа, что это будет…

Предполагаю, что текст программы будет составляться на стороне сервера, на сервере пользователь будет просто выбирать тип программы, далее «набирать» программу с помощью галочек, кнопочек, по сути конструктора программ.

Сервер не будет полностью составлять С код программы, он и не сможет это правильно сделать, насколько я знаю, STM32 поделен на шины, иногда чтобы включить вход такой то – надо сначала подать питание на шину, если оно не подано… Плюс, подозреваю, что из-за общей сложности контроллера – не стоит позволять писать любой код программы, мы потом замучаемся его отлаживать и искать проблемы… предлагаю в контроллере просто предусмотреть ряд функций, для включения/выключения выходов в зависимости от переданных в них параметров, а в этих функциях уже смотреть, что ж за параметры были переданы, надо ли включать питание шины, можно ли вообще включать данных выход (вдруг он уже задействован для какой-то другой цели и т.д.)

Типы программ будут такие:

1. Включить/выключить 1 цифровой выход если цифровой вход = значению

Прототип функции set\_io(“vhod1”,0,”vuhod2”,”1”) –если vhod1=0 –установить выход 2 в 1

Тут только 4 значения – контролируемый цифровой вход, его значение, цифровой выход, его значение

1. Включить/выключить 1 цифровой выход если аналоговый вход в интервале значений

Прототип функции set\_io(“analog\_vhod1”,0,120,”vuhod2”,”1”) - указываем аналоговый вход, диапазон значений, и цифровой выход, его значение при соблюдении условий

1. Установить значение в аналоговый выход если цифровой вход…

Прототип функции set\_analog (“vhod1”,0,”analog\_vuhod2”,”125”) –если vhod1=0 –установить аналоговый выход 2 в 125

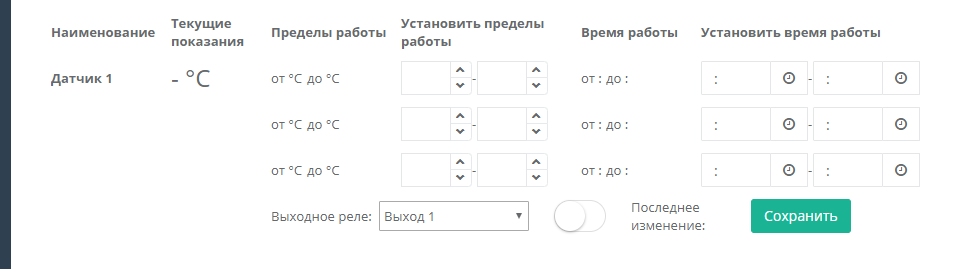
1. Установить значение в аналоговый выход если аналоговый вход…

Прототип функции set\_analog (“analog\_vhod1”,0,125,”analog\_vuhod2”,”125”) –если аналоговый вход имеет значение в интервале 0, 125–установить аналоговый выход 2 в 125

1. И так далее предлагаю, например два входа, один выход, один вход, три выхода…. Определится нужно будет с максимальным числом таких вариаций, ограничить их и установить правило, например не более трех входов на контроль, не более трех выходов на воздействие… надо в этом плане обсудить с заказчиком и всеми нам, какой вариант используем

## Температурные профили

Еще у нас есть такая штука, как температурные профили….



## Задание температурных профилей

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”set\_temp\_profile”,

“value”:”

{

“sensor”:”1”,номертемпературногодатчика

“relay”:1, номер выхода (цифрового, реле)

“enable”:”1”,включен или выключен профиль

“period1”:”{“start\_temp”:”20”,”end\_temp”:”35”,”start\_time”:”00:25”,”end\_time”:”12:00”}”, начальная температура, конечная температура, начальное время, конечное время

“period2”:”{“start\_temp”:”20”,”end\_temp”:”35”,”start\_time”:”12:25”,”end\_time”:”14:00”}”,

“period3”:”{“start\_temp”:”20”,”end\_temp”:”35”,”start\_time”:”14:25”,”end\_time”:”23:00”}”

}

}”,

“time”:1566682245

}

Температурные профили привязываются к конкретному температурному датчику.

Периодически контроллер будет считывать уже установленные температурные профили:

## Считывание температурных профилей

Сервер считывает температурные профили следующей командой:

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”get\_temp\_profiles”,

“value”:”1” - номер сенсора, с которого данные о температурном профиле нужно передать

}”,

“time”:1566682245

}

Ответом от контроллера будет:

…………………

“temp\_sensors”: [24,255,145,12….]– температурные сенсоры, массив значений

} ,

“command\_result”:

”{

“type”:”get\_temp\_profiles”, - повторяется тип команды, на которую мы отвечаем

“value”:”

{

“sensor”:”1”, номертемпературногодатчика

“relay”:1, номер выхода (цифрового, реле)

“enable”:”1”,включен или выключен профиль

“period1”:”{“start\_temp”:”20”,”end\_temp”:”35”,”start\_time”:”00:25”,”end\_time”:”12:00”}”, начальная температура, конечная температура, начальное время, конечное время

“period2”:”{“start\_temp”:”20”,”end\_temp”:”35”,”start\_time”:”12:25”,”end\_time”:”14:00”}”,

“period3”:”{“start\_temp”:”20”,”end\_temp”:”35”,”start\_time”:”14:25”,”end\_time”:”23:00”}”

}

}”,

“firmware\_id”:”100”,

…………………………….

Сервер просто знает, что у нас 8 температурных датчиков, и собственно перебирает все датчики и считывает с них профили.

## Удаленная прошивка

Если версия прошивки контроллера меньше, чем текущая актуальная прошивка, указанная на сервере, тогда сервер никаких других команд контроллеру не присылает, кроме одной:

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”burn”,

“value”:”105” - номер прошивки, которую нужно загрузить контроллеру

}”,

“time”:1566682245

}

Контроллер после этого отправляет на сервер уже такой пакет:

“temp\_sensors”: [24,255,145,12….]– температурные сенсоры, массив значений

} ,

“command\_result”:

”{

“type”:”get\_update”, - повторяется тип команды, на которую мы отвечаем

“value”:”105”, номер прошивки, которую контроллер хочет загрузить

}

“firmware\_id”:”100”,

И в ответ сервер просто пришлет нужную прошивку в бинарном виде

## Управление и смена ключей шифрования.

Подразумевается, что у нас будет два типа ключей шифрования – сессионные и бэкдор-ключи.

Бэкдор-ключи будут у всех контроллеров одинаковые, абсолютно у всех, железно записанные в прошивку. Сессионные ключи приходят с сервера и устанавливаются контроллером в память, и все следующие пакеты шифруются уже сессионным ключом…

Первый пакет (если сессионного ключа нет, контроллер или свежий, или ключ не сохранен – потеря питания, еще какая то проблема) шифруется бэкдор ключом.

Этот первый пакет будет также передавать все параметры, как и обычный пакет, но в поле command\_result будет уже всегда одно значение

{

“id”:”AFHDCBSDSJKHSA8934743”, - Любая цифро-буквенная комбинация, уникальный ид контроллера,

“state”: {

“inputs”: [0, 0, 1,1,0,1,0,1…..], - цифровые входы, массив значений, 0-1

“analog\_inputs”: [251,25,125,0,…..], - аналоговые входы, массив значений

“relays”:[0, 1, 1…..], - цифровые выходы, реле, массив значений, 0-1

“analog\_outputs”: “[14,152,0,255…]”, – аналоговые выходы, текущее состояние, массив значений

“temp\_sensors”: [24,255,145,12….] – температурные сенсоры, массив значений

} ,

“command\_result”:”need\_keys”

“firmware\_id”:”100”, - ид прошивки, числовое значение

“time”:1566682245 – текущее время контроллера, unixtimestamp

}

Сервер сначала проверит номер прошивки, если нужно – пришлет команду на обновление (все пакеты при этом дешфируются бэкдор ключами, в том числе и сам бинарный файл прошивки), если с номером прошивки все нормально, ответом сервера будет:

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”need\_keys”,

“value\_key”:”A5B554D2C122….”, – ключ в шестнадцатеричном формате

“value\_iv”:”D5B554D2C122….” – значение вектора инициализации

}”,

“time”:1566682245

}

Все остальные пакеты контроллер должен слать уже шифрованные этим ключом.

Логика простая, если сессионных ключей нет – значит ключ шифрования – это бэкдоровский ключ. И все шифруется и дешифруется им (и файл прошивки в том числе). Если сессионный ключ есть – значит ключ шифрования – это сессионный ключ, и все шифруется и дешифруется им (и файл прошивки в том числе)

## Синхронизация времени

При необходимости синхронизировать время от сервера придет такой пакет:

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”set\_time”,

“value”:”1652545645” – время в формате unixtimestamp

}”,

“time”:1566682245

}

Это время не то, что используется при передаче пакетов! При передаче пакетов время (поле time) – это скорее больше счетчик, чем время…а вот синхронизация времени на контроллере – это уже то время, которое будет у него на дисплее, которое будет использоваться в температурных профилях и т.д.

Ответом на эту команду будет:

{

“id”:”AFHDCBSDSJKHSA8934743”, - Любая цифро-буквенная комбинация, уникальный ид контроллера,

“state”: {

“inputs”: [0, 0, 1,1,0,1,0,1…..], - цифровые входы, массив значений, 0-1

“analog\_inputs”: [251,25,125,0,…..], - аналоговые входы, массив значений

“relays”:[0, 1, 1…..], - цифровые выходы, реле, массив значений, 0-1

“analog\_outputs”: “[14,152,0,255…]”, – аналоговые выходы, текущее состояние, массив значений

“temp\_sensors”: [24,255,145,12….] – температурные сенсоры, массив значений

} ,

“command\_result”:

”{

“type”:”set\_time ”, - повторяется тип команды, на которую мы отвечаем

“value”:”1652545645”, время в формате timestamp

}

“firmware\_id”:”100”, - идпрошивки, числовоезначение

“time”:1566682245 – текущеевремяконтроллера, unixtimestamp

}

## Запрос времени от сервера

Сервер может и спросить контроллер, какое время у него сейчас установлено:

{

“result”:”success”, - метка result, будет в каждом ответе

“command”:

”{

“type”:”get\_time”,

“value”:””

}”,

“time”:1566682245

}

Ответом будет:

{

“id”:”AFHDCBSDSJKHSA8934743”, - Любая цифро-буквенная комбинация, уникальный ид контроллера,

“state”: {

“inputs”: [0, 0, 1,1,0,1,0,1…..], - цифровые входы, массив значений, 0-1

“analog\_inputs”: [251,25,125,0,…..], - аналоговые входы, массив значений

“relays”:[0, 1, 1…..], - цифровые выходы, реле, массив значений, 0-1

“analog\_outputs”: “[14,152,0,255…]”, – аналоговые выходы, текущее состояние, массив значений

“temp\_sensors”: [24,255,145,12….] – температурные сенсоры, массив значений

} ,

“command\_result”:

”{

“type”:”get\_time ”, - повторяется тип команды, на которую мы отвечаем

“value”:”1652545645”, время в формате timestamp

}

“firmware\_id”:”100”, - идпрошивки, числовоезначение

“time”:1566682245 – текущеевремяконтроллера, unixtimestamp

}

Это пока основные моменты, сознательно опускаем внутреннюю память, ее очистку, чтение из него и ряд других вспомогательных моментов.

Жду ваших комментариев и предложений, протокол пока не аксиома, я буду его реализовывать в течение этих дней в рабочий, но его нужно будет еще обсуждать и корректировать