Ненавязчивая автоматизация ректификационной установки

Вариант LITE, v.041x

По мотивам ветки «Ненавязчивая автоматизация ректификационной установки» на российском форуме ХоумДистиллер

Оглавление

1. Введение

Когда винокур начинает заниматься ректификацией спиртосодержащего сырья, рано или поздно он приходит к мысли о необходимости автоматизации этого процесса. Или - хотя бы некоторых базовых операций, таких как вывод колонны на рабочий режим, автоматическое регулирование мощности нагрева куба, скорости отбора на разных этапах ректификации и т.п. В настоящее время в Сети описано большое количество разнообразных систем автоматизации процесса ректификации. Но, так или иначе, любая такая система состоит из 1) исполнительных устройств (нагреватели, регуляторы скорости отбора и т.п.), 2) разнообразных датчиков (температуры, давления, напряжения питающей сети и т.п.) и 3) управляющего компьютера (или микроконтроллера) с соответствующим программным обеспечением, которые занимаются сбором информации с датчиков и управлением исполнительными устройствами. Разница между системами автоматизации, по сути, заключается лишь в организации взаимодействия этих подсистем и «распределении обязанностей» между ними.

Довольно часто система автоматизации представляют собой некий «моноблок» в основе которого лежит микроконтроллер (а иногда и несколько), обеспечивающий весь функционал системы. От низкоуровневого управления исполнительными устройствами и сбора данных с датчиков до реализации общей логики управления процессом и пользовательского интерфейса. В таких системах обычно предусмотрены средства изменения конфигурации оборудования и поведения системы. Но часто эти возможности ограничены из-за их «параметрического» (количественного) характера. Модификации производятся лишь путем изменения параметров, таких как количества температурных датчиков, устройств отбора, или же параметров самого процесса. Добавление же новых типов устройств или существенное изменение логики управления обычно связаны с перепрограммированием микроконтроллеров, что может представлять серьезные трудности для пользователя, не являющегося разработчиком данного программного обеспечения. Основная причина заключается в том, что программирование микроконтроллеров осуществляется, как правило, на низкоуровневых языках программирования (C, C++), а модификация чужого низкоуровневого кода требует достаточно высокой квалификации программиста и глубокого «погружения» в логику работы приложения.

В настоящее время наблюдается заметный рост популярности (главным образом - за счет платформы Arduino) и существенное падение цен на 8-битные микроконтроллеры. Это дает возможность снабдить каждое исполнительное устройство или датчик своим микроконтроллером, освободив тем самым управляющий компьютер от непосредственного выполнения низкоуровневых операций, связанных с данным «железом». Кроме того, такая «интеллектуализация» первичного «железа» дает возможность автоматического обнаружения, идентификации, тестирования подключенного к системе оборудования и создание соответствующей программной инфраструктуры для управляющего компьютера.

Будет уместно отметить еще один фактор, облегчающий решение отмеченных выше проблем традиционных схем автоматизации. Это появление недорогих, миниатюрных, но достаточно мощных микрокомпьютеров, снабженных полноценными linux-подобными операционными системами (один из наиболее популярных примеров — знаменитая Raspberry Pi, в миру - «малинка»). В результате появляется возможность реализации пользовательского интерфейса и общей логики управления процессом на высокоуровневых языках программирования, таких как простой и очень легкий в изучении язык python.

С учетом указанных факторов, систему автоматизации удобно (и вполне естественно) разбить на два уровня. Первый уровень - это уровень «реального времени». Он состоит из набора модулей, каждый которых содержит датчик или исполнительное устройство и микроконтроллер, обеспечивающий низкоуровневые операции в реальном времени и обмен данными со вторым уровнем. А на втором уровне находится управляющий микрокомпьютер с операционной системой и высокоуровневыми языками программирования, занимающийся задачами не требующими реакций в реальном времени, такими как общее управление процессом, пользовательский интерфейс, протоколирование и т.п.

Ниже, на примере классической ректификационной установки, описывается вариант построения системы автоматизации, основанной именно на таких принципах.

2. Первый уровень - аппаратная часть

Как уже было отмечено во введении, аппаратная часть автоматики имеет модульную структуру. Каждый модуль содержит микроконтроллер, который, с одной стороны, обеспечивает функционирование датчиков и исполнительных устройств, оцифровкой аналоговых сигналов или обменом данными с цифровыми устройствами, а с другой стороны — обменом с управляющим компьютером второго уровня. Удобным, гибким и вполне естественным способом физического объединения этих модулей является крейт со сквозными общими сигнальными и силовыми линиями. С него и начнем.

2.1. Крейт

Блок-схема крейта представлена на рис.1. К сигнальной шине крейта подключен управляющий компьютер Raspberry Pi 3 («малинка»), питание которого производится от отдельного блока питания, подключенного к устройству бесперебойного питания (UPS, на рисунке не показан).

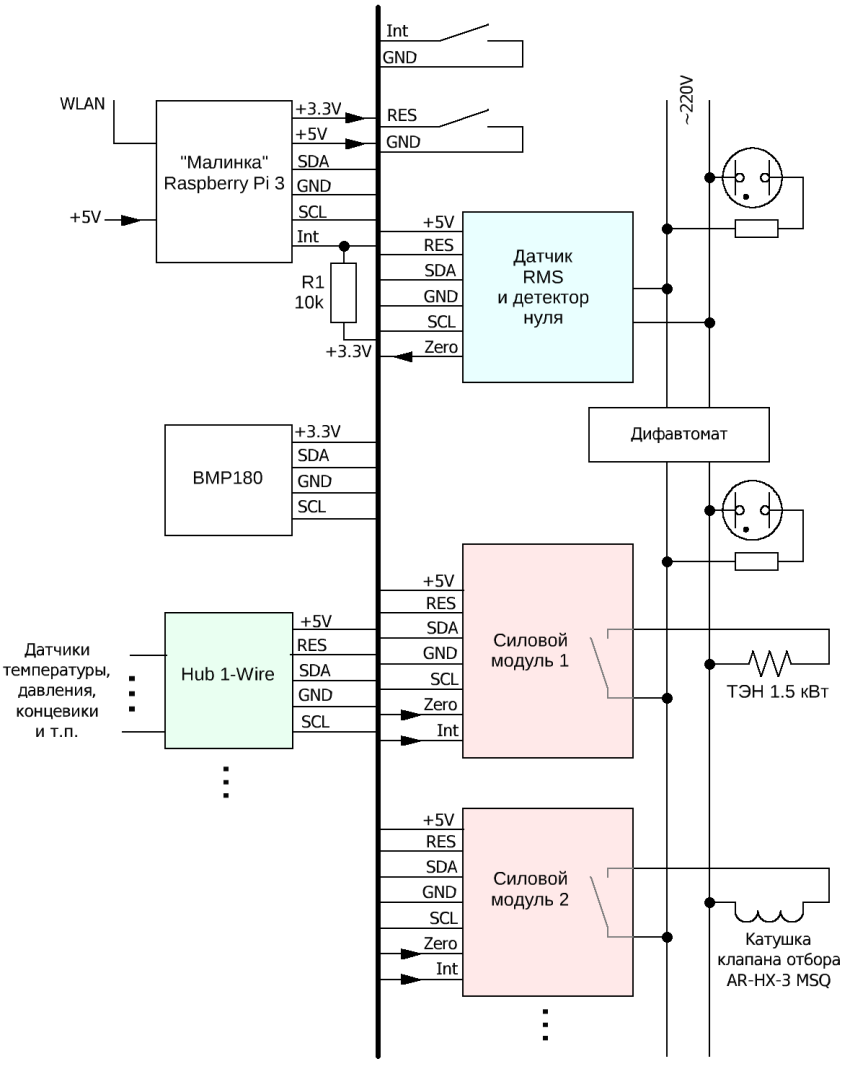

Рис.1. Блок-схема крейта

Обмен данными между малинкой и модулями осуществляется по шине I2C на уровне 3.3В (используются подтягивающие резистора самой малинки). Поэтому никакие согласующие устройства для 5-вольтовых микроконтроллеров модулей не требуется. Питание модулей +3.3V и +5V производится непосредственно от малинки.

Датчики, имеющие интерфейс I2C (например, датчик атмосферного давления BMP180) подключаются непосредственно к шинам крейта. Датчики с интерфейсом 1-Wire (например, датчики температуры DS18B20) подключаются через хабы 1-Wire, выполненные на базе тех же микроконтроллеров, что и другие модули. Аналоговые датчики (например, дифференциальный датчик кубового давления MPX5010DP) подключаются непосредственно к АЦП отдельных микроконтроллеров, которые осуществляют накопление и предварительную обработку данных. На этих же микроконтроллерах реализованы цифровые интерфейсы 1-Wire (как, например, для датчика кубового давления) или I2C (как, например, для датчика RMS) для подключения к 1-Wire хабу или непосредственно к общей шине крейта соответственно.

Более детально сигнальная часть общей шины крейта показана на рис.2. По линии Zero идут импульсы детектора нуля сетевого напряжения (физически детектор нуля объединен с датчиком RMS). Эти сигналы используются для синхронизации силовых модулей с питающей сетью. Линия Int предназначена для инициализации аппаратного прерывания для всех модулей одновременно. Через линию RES производится подача низкого уровня на вход RESET всех микроконтроллеров модулей инициируя их перезагрузку.

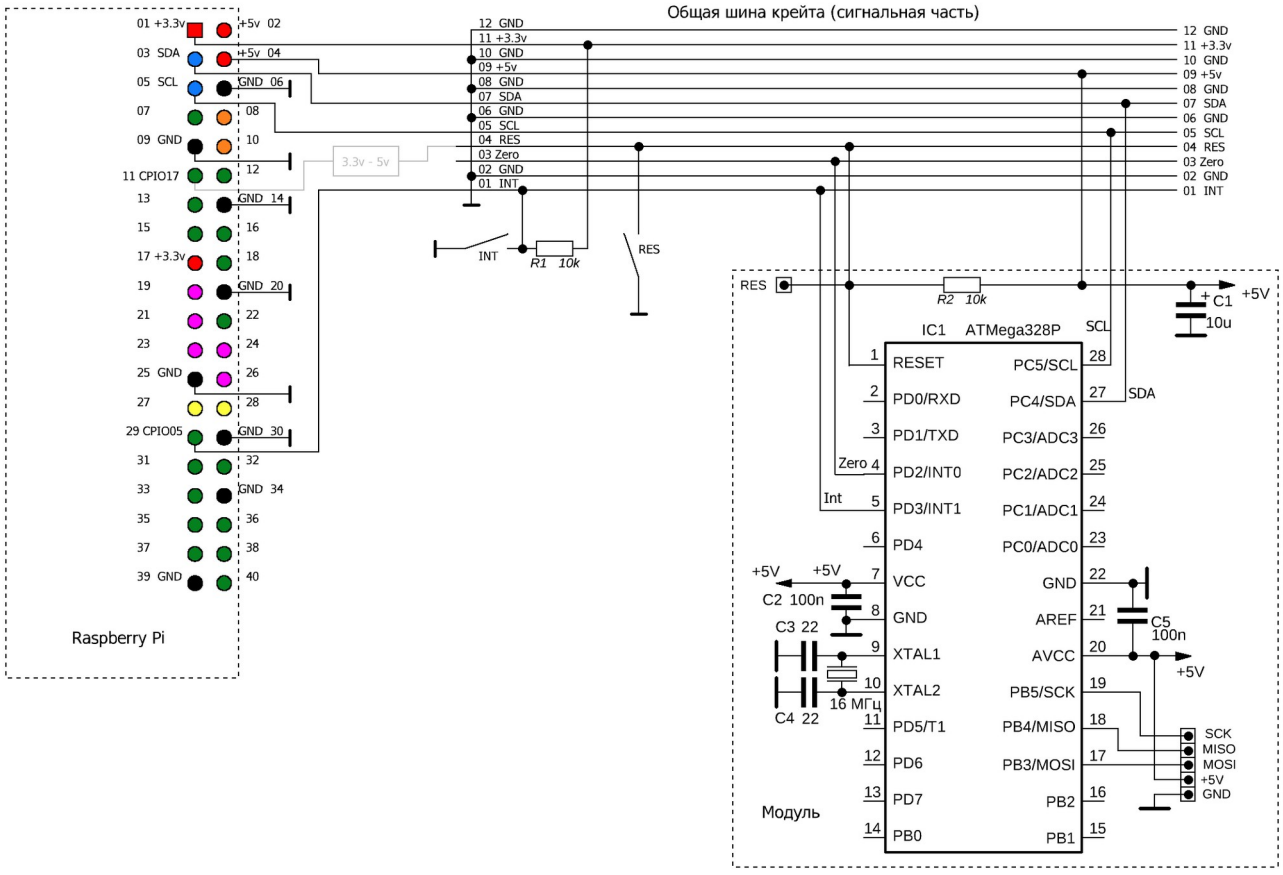

Рис.2. Сигнальная часть общей шины крейта.

На этом же рис.2 внизу справа показана схема подключения микроконтроллеров модулей к общей шине. Эта часть унифицирована для все модулей.

Конструктивно крейт выполнен на единой пластине из стеклотекстолита (см. рис.3). Малинка расположена слева. С низковольтной частью общей шины она связана гибкими проводниками подсоединенными к одному из слотов шины.

Низковольтная часть общей шины выполнена на отдельной платине фольгированного стектотекстолита с распаянными слотами (на рис.3 спереди в центре). Соответствующий lay-файл находится в файле 03_crate_low_voltage_bus_pcb.lay6, а рисунок платы — в файле 03_crate_bus_pcb.gif в папке lays в архиве приложения к данному документу.

Высоковольтная часть общей шины выполнена из отрезков латунных шин. Здесь же в центре сзади находится и общий радиатор для симисторов силовых модулей. Справа на кусочке DIN-рейки расположен дифавтомат и индикаторы напряжения (в сети и на шине).

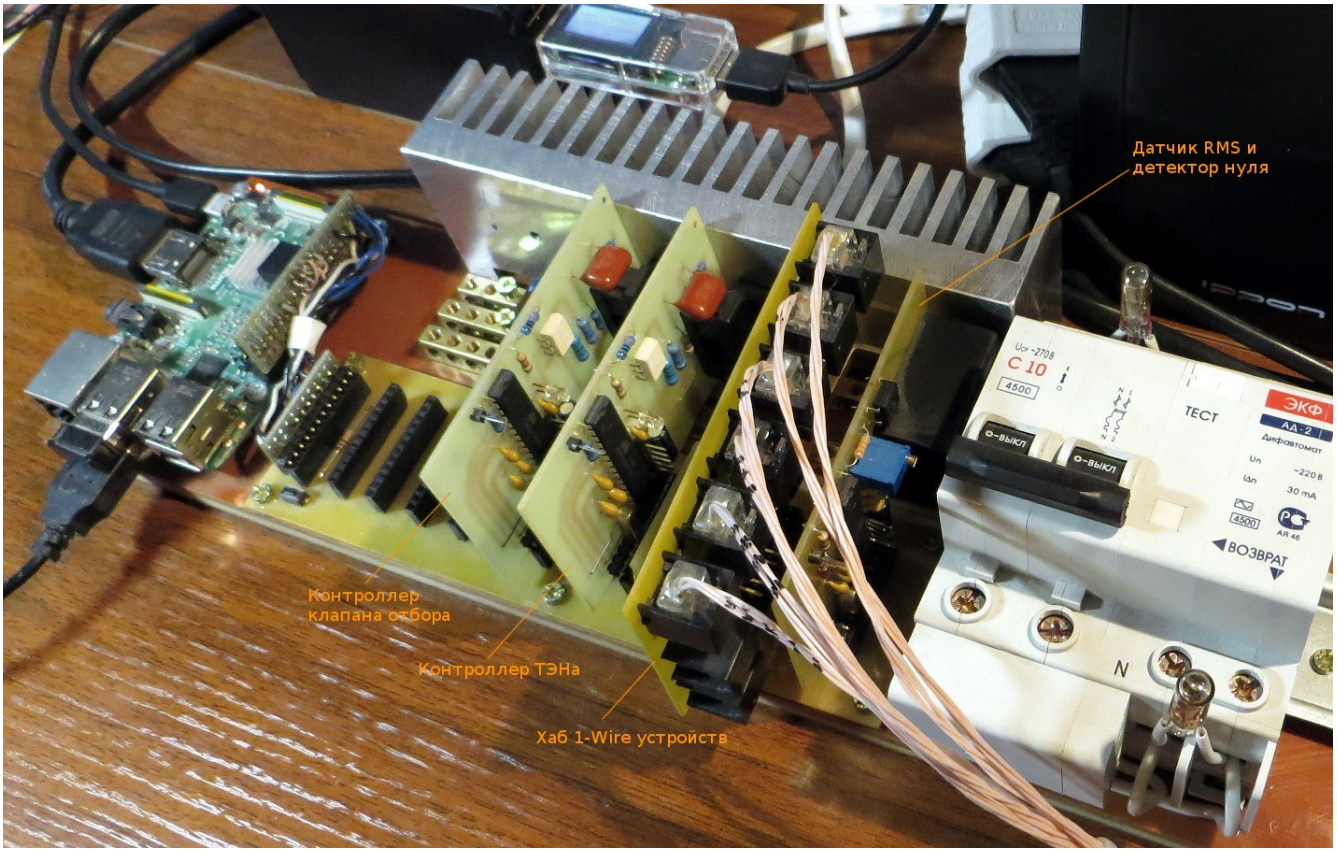

Рис.3. Вариант конструктивного исполнения крейта.

На рис.3. изображены также и несколько модулей, подключенных к слотам крейта. Фактически это минимальный набор модулей, позволяющий управлять процессом кубовой ректификации. Это (справа налево): 1) датчик RMS сетевого напряжения с детектором нуля, 2) один хаб для 1-Wire датчиков DS18B20, 3) контроллер ТЭН-а и 4) контроллер клапана отбора. Эти и некоторые другие модули рассмотрены ниже. Небольшая платка с датчиком атмосферного давления BMP180 с разъемом для подключения к слоту, «затерялась»между контроллером ТЭНа и хабом 1-Wire.

2.2. Датчик RMS сетевого напряжения с детектором нуля

Назначение датчика RMS — измерение среднеквадратичного напряжения питающей сети. Этот параметр необходим для коррекции мощности нагрева при больших колебаниях сетевого напряжения. Помимо измерения RMS, микроконтроллер формирует короткие импульсы на шине Zero, задние фронты которых совпадает с моментами прохождения сетевого напряжения через ноль. Эти сигналы необходимы для синхронизации работы некоторых силовых модулей (контроллеры PWM и PDM) с питающей сетью.

Схема датчика представлена на рис.4. Сетевое напряжение понижается при помощи трансформатора Tr1 и выпрямляется мостом DB107 (IC2). Это напряжение через ограничивающий резистор и стабилитрон D1 подается на вход аппаратного прерывания микроконтроллера INT0 для формирования импульсов детектора нуля на линии Zero сигнальной шины крейта.

Параллельно, это же напряжение с моста IC2 через делитель R3-R4 подается на канал 0 АЦП микроконтроллера, оцифровывается и накапливается для усреднения. Период запуска АЦП — 200 мкс. Весь цикл измерения RMS длится 1 сек. За это время в микроконтроллере накапливается 5000 отсчетов квадратов напряжения сети, по которым производится усреднение.

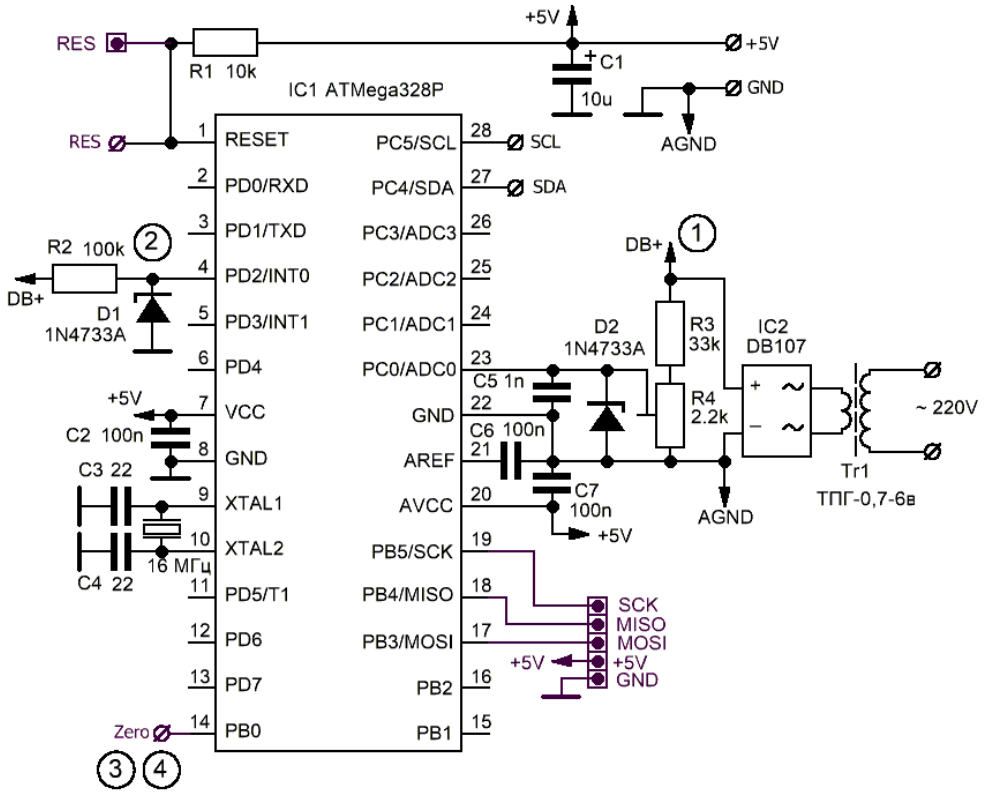

Рис.4. Схема модуля датчика RMS с детектором нуля.

Вариант разводки в виде lay-файла (файл 04_rms_18.01.11.2_pcb_07.lay6) и рисунок платы (файл 04_rms_18.01.11.2_pbc_07.gif) находятся в папке lays архива приложения. Фотография готовой платы представлена на рис.4.

Исходный файл прошивки (main.c) и hex-файл (main.hex) находятся в архиве приложения в папке firmware/A1_RMS. Как выполнить прошивку микроконтроллера модуля описано в разделе 2.7. «Прошивка» микроконтроллеров модулей.

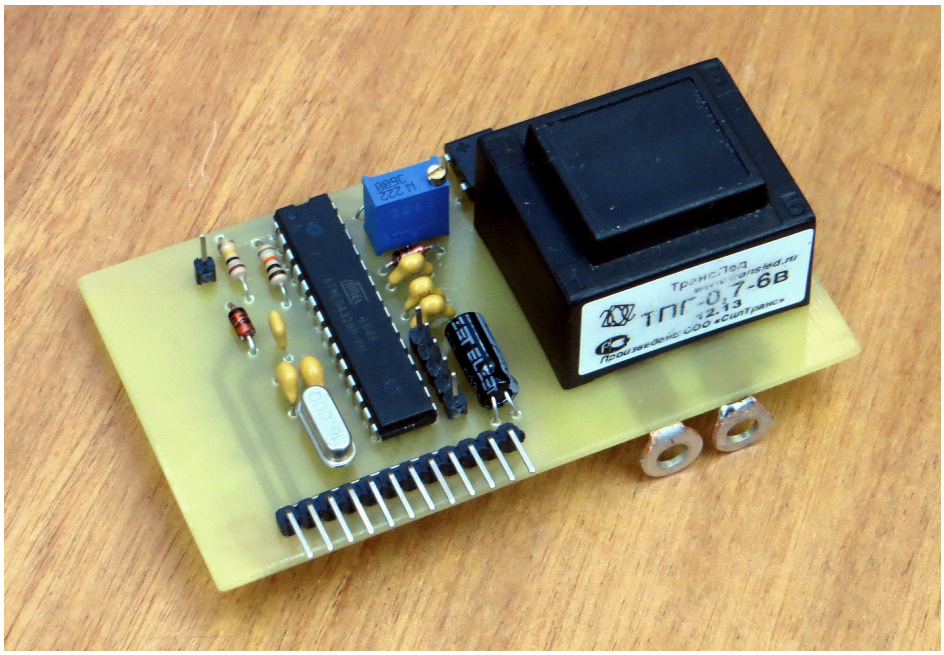

Рис.5. Готовая плата модуля датчика RMS с детектором нуля.

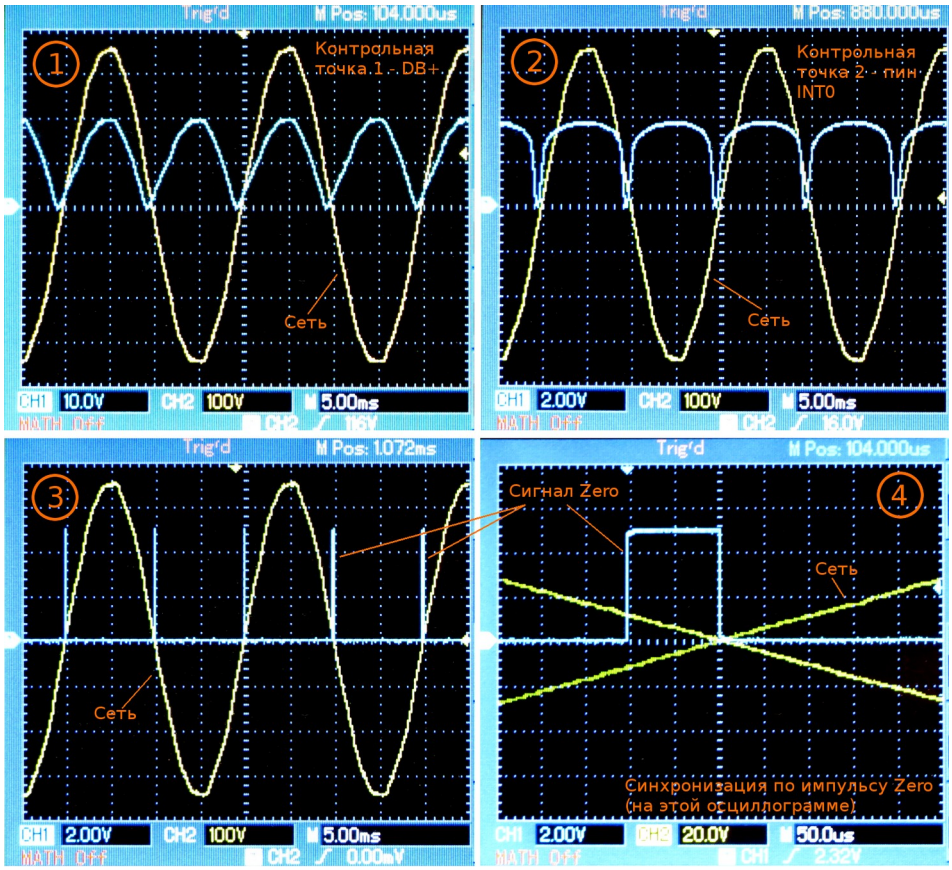

Рис.6. Осциллограммы в контрольных точках, указанных на схеме рис.4.

Если монтаж и прошивка микроконтроллера выполнены без ошибок, то никакой дополнительной отладки не требуется. Если же что-то пошло не так, то осциллограммы в контрольных точках, обозначенных на рис.3 окружностями с номерами, показаны на рис.6. Они могут помочь в поиске неисправностей.

2.3. Силовой модуль

Силовые модули предназначены для управления исполнительными устройствами, такими как ТЭНы, клапаны отбора, контакторы и т. п. Схемы всех силовых модулей одинаковы. Они могут отличаться лишь номиналами RC-цепочки, включенной параллельно симистору, и прошивками микроконтроллеров.

Схема силового модуля представлена на рис.7. Вход драйвера симистора MOC3083 через ограничивающий резистор R5 подключен к пину PD0 микроконтроллера модуля. Драйвер имеет оптическую развязку и детектор нуля. В результате включение симистора BTA16 происходит в нуле сетевого напряжения. Это существенно снижает уровень помех, но в нагрузку может быть передано только целое количество сетевых полупериодов, что может иногда накладывать некоторые (как правило, несущественные) ограничения на точность регулирования параметров исполнительных устройств.

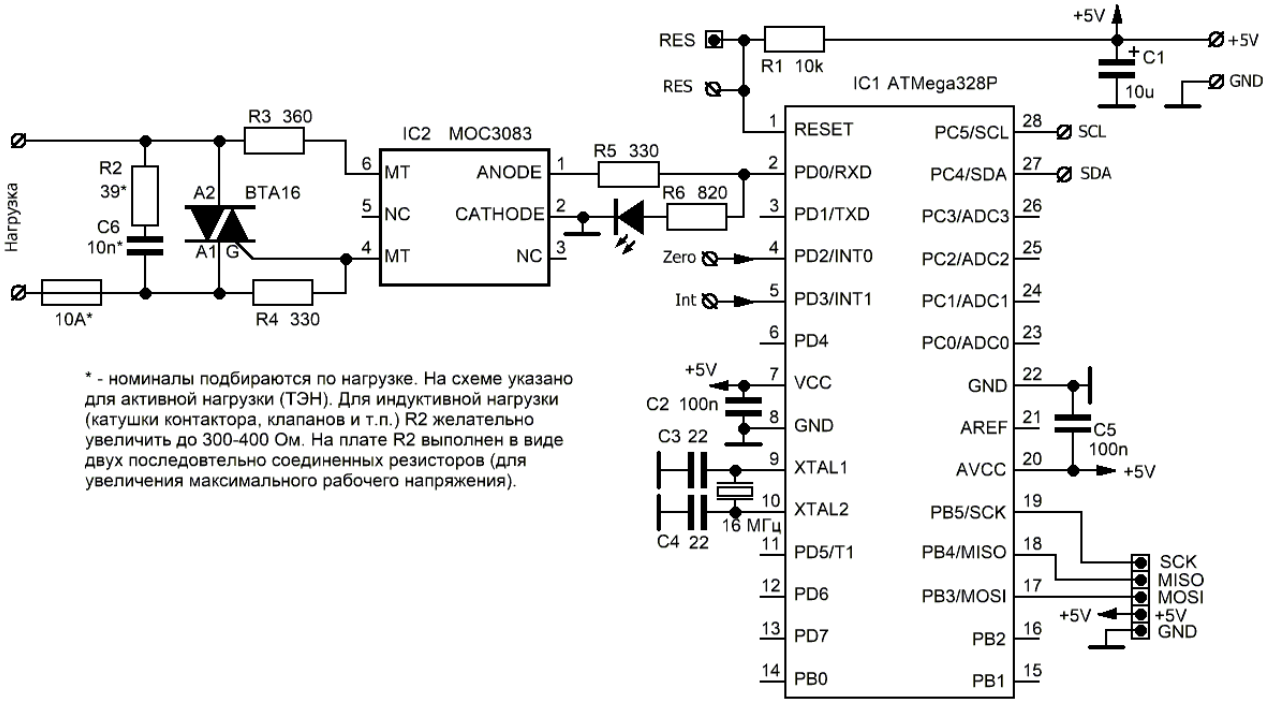

Рис.7. Схема силового модуля.

Схема включения симистора и драйвера стандартная, согласно datasheet на микросхему MOC3083. Поэтому в комментариях не нуждается. К этому же пину (PD0) микроконтроллера через резистор R6 подключен светодиод. Это очень удобно для визуального контроля работы устройства. Как во время отладки, так и во время реальной работы установки.

Вариант разводки в виде lay-файла (файл 07_pm_18.01.15.2_pcb_02.lay6) и рисунок платы (файл 07_pm_18.01.15.2_pcb_02.gif) находятся в папке lays архива приложения. Фотография готовой платы представлена на рис.8.

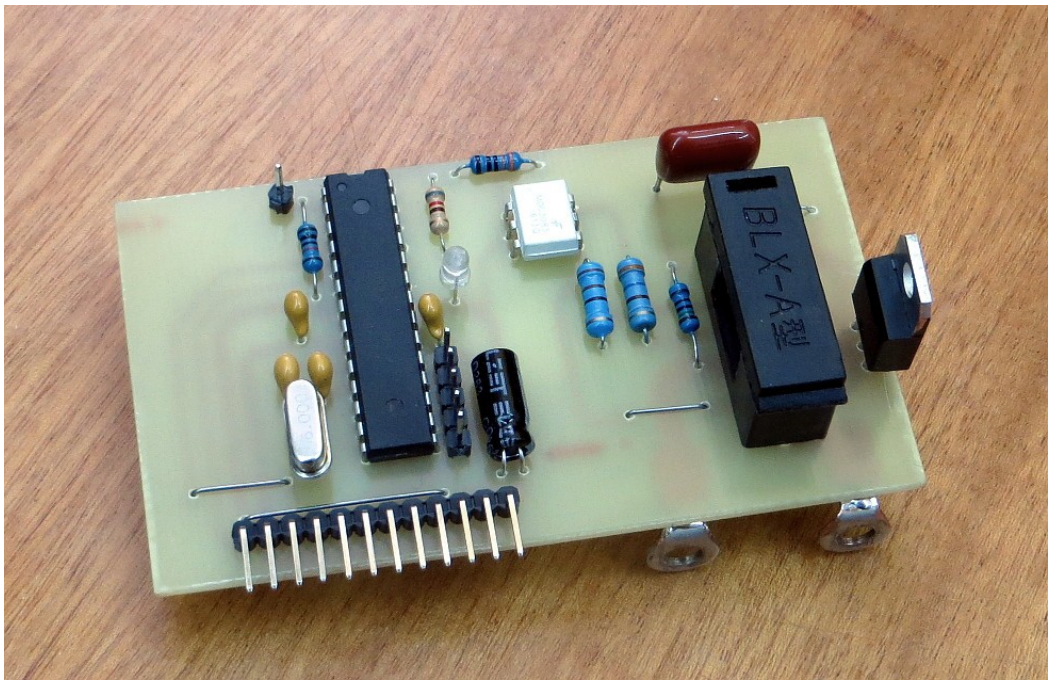

Рис.8. Готовая плата силового модуля.

Для силовых модулей разработаны три варианта прошивок в зависимости от того, каким исполнительным устройством данный модуль управляет.

1. **Релейный модуль** предназначен для управления исполнительными устройствами, имеющими два состояния (включено и выключено). Примером такого устройства является контактор. Исходный файл прошивки (main.c) и соответствующий hex-файл (main.hex) находятся в папке firmware/A2_relay архива приложения.

2. **PDM — модуль** используется для регулирования мощности инерционных нагрузок, таки как ТЭНы. PDM это аббревиатура Pulse Density Modulation. Такой способ модуляции позволяет пропускать к нагрузке сетевые полупериоды так, чтобы отклонение средней мощности от заданного уровня было минимальным. Такой способ часто еще называют методом коррекции ошибок или методом Брезенхема, широко используемым в растровой компьютерной графике. В данной автоматике силовой модуль, реализующий такой способ модуляции сетевых полупериодов, используется для регулирования мощности нагрева ТЭНа. Исходный файл прошивки (main.c) и соответствующий hex-файл (main.hex) находятся в папке firmware/A3_PDM архива приложения. Подробнее почитать о таком способе модуляции можно в топике https://forum.homedistiller.ru/msg.php?id=12906068 на форуме.

3. **PWM — модуль** в данной автоматике используется для регулирования скорости отбора путем регулирования относительной длительности периода времени, в течение которого сетевые полупериоды пропускаются к нагрузке. PWM — это английская аббревиатура широтно-импульсной модуляции (ШИМ). В данном регуляторе частота ШИМ составляет 0.1Гц (период - 10 сек). Поэтому точность регулирования скорости отбора составляет 0.1%. Исходный файл

прошивки (main.c) и соответствующий hex-файл (main.hex) находятся в папке firmware/A4_PWM архива приложения.

Как выполнить прошивку микроконтроллера модуля описано в разделе 2.7. «Прошивка» микроконтроллеров модулей

2.4. Драйвер сервомашинки MG90S

Драйвер сервомашинки по сути представляет собой управляемый по шине I2C генератор ШИМ-сигнала с дополнительным источником питания сервопривода. Схема его представлена на рис.9 (источник питания сервопривода не показан). Схема простейшая — стандартная «обвязка» микроконтроллера. Поскольку весь функционал драйвера реализован средствами самого микроконтроллера, никаких дополнительных компонентов, кроме блока питания двигателя не требуется. ШИМ-сигнал управления сервоприводом берется непосредственно с пина PB1 микроконтроллера.

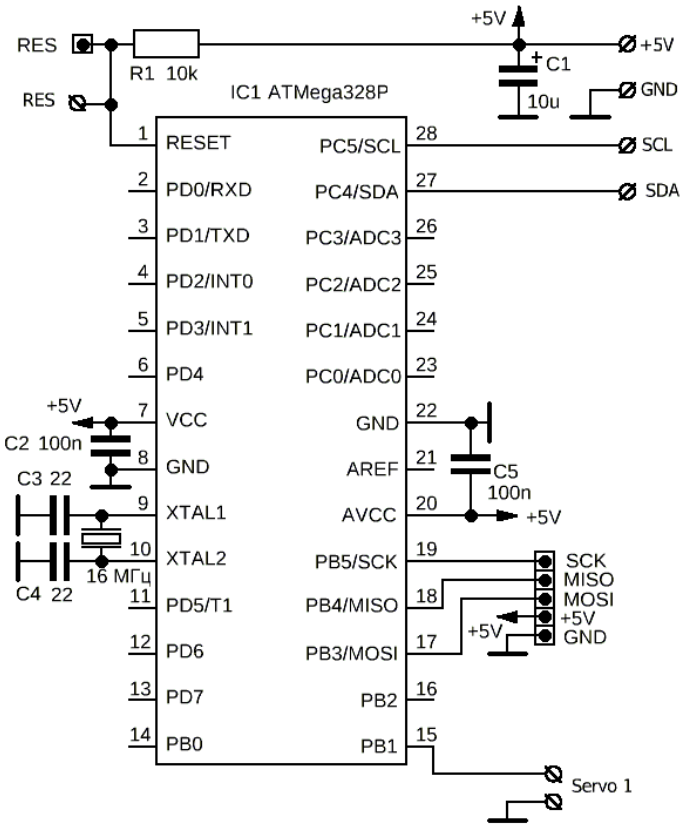

Рис.9. Схема драйвера сервопривода.

Вариант разводки в виде lay-файла (файл 09_servo_19.10.05_pbc_02.lay6) и рисунок платы (файл 09_servo_19.10.05_pbc_02.gif) находятся в папке lays архива приложения.

Сервомашинки под нагрузкой могут потреблять значительный ток. Поскольку питание 5-вольтовой линии крейта производится от малинки, то питание сервомашинки желательно обеспечить от отдельного источника питания. Кстати, это даст возможность подключать к

контроллеру другие двигатели, отличающиеся мощностью или напряжением питания. В данном случае, я просто «прилепил» к плате контроллера небольшую покупную плату сетевого блока питания на 5В 1А, подключив его непосредственно к сетевым шинам крейта. То что получилось показано на следующем рис.10. Как это все разместилось в крейте показано на рис.11.

Исходный файл прошивки (main.c) и соответствующий hex-файл (main.hex) находятся в папке firmware/A9_servo архива приложения. Как выполнить прошивку микроконтроллера модуля описано в разделе 2.7. «Прошивка» микроконтроллеров модулей.

В данной автоматизации этот контроллер сервомашинки используется для фракционирования отбираемого спирта (т. е. - для переключения потока на разные приемные емкости). Описание самого исполнительного устройства (фракционника) можно посмотреть в топике https://forum.homedistiller.ru/msg.php?id=13780113 на форуме. Там же есть stl-файлы для 3D печати его компонентов.

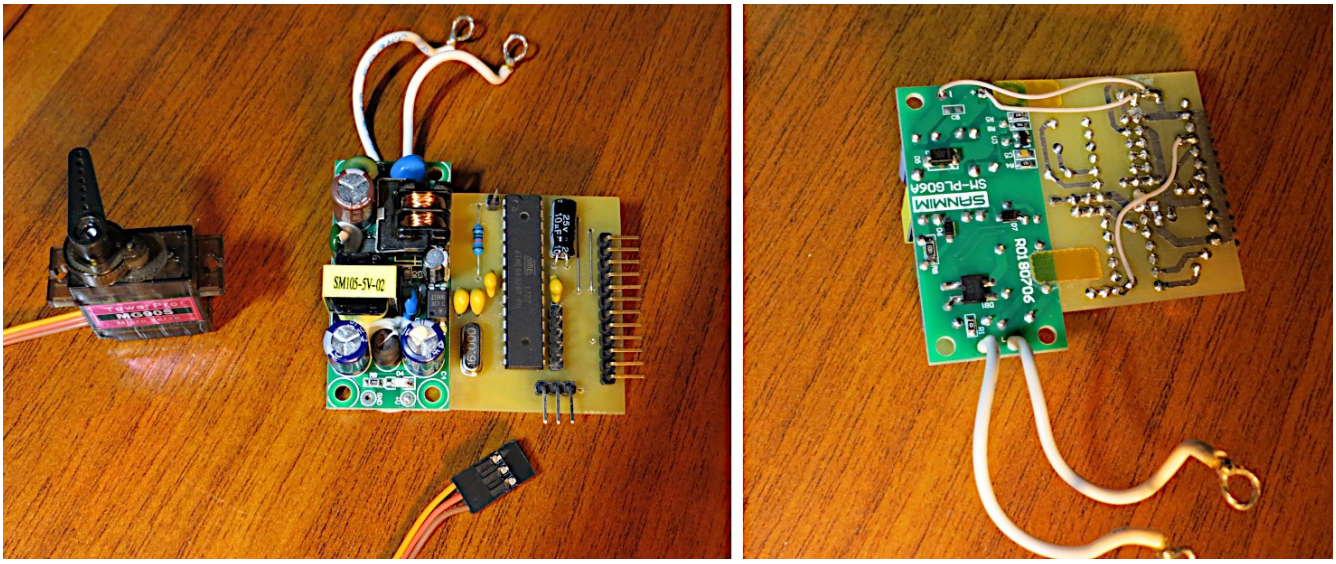

Рис.10. Готовая плата контроллера с дополнительным блоком питания сервопривода.

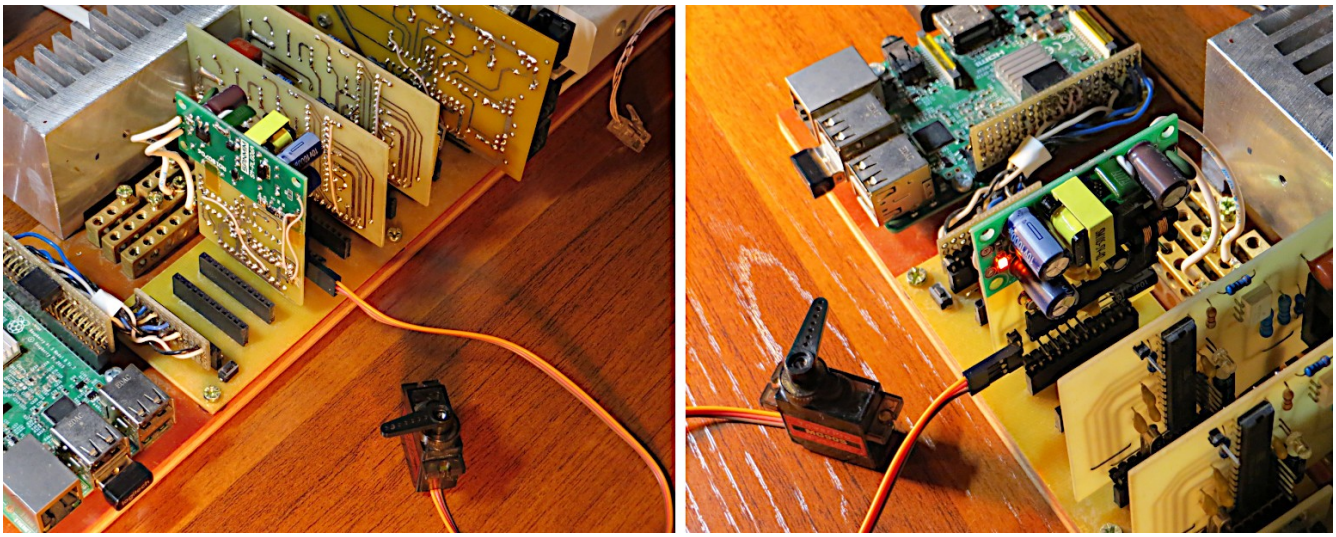

Рис.11. Плата контроллера сервомашинки в крейте.

2.5. Хаб 1-Wire устройств

В принципе, малинка поддерживает шину 1-Wire, к которой можно подключить целую гирлянду 1-Wire устройств, например, таких как популярные датчики температуры DS18B20. Однако при таком подходе возникает несколько неприятных моментов. Во-первых, у малинки эта шина 3.3-вольтовая, что несколько снижает помехоустойчивость шины. Во-вторых, при сбое одного устройства блокируется вся шина. Ну и в-третьих, при размещении всех датчиков на одной шине считывание данных с них можно производить только последовательно. При большом количестве датчиков опрос всех устройств может занять заметное время. Поэтому в данной системе применен другой подход для работы с 1-Wire устройствами.

В основе 1-Wire хаба опять-таки лежит микроконтроллер ATMega328P с минимальной "оснасткой". Каждый пин порта D микроконтроллера реализует виртуальную 1-Wire шину, без адресации. Поэтому к каждой такой 1-Wire шине может быть подключено только одно 1-Wire устройство, но зато все устройства, подключенные к одному хабу могут работать параллельно и синхронно. В том числе и считывание данных с них.

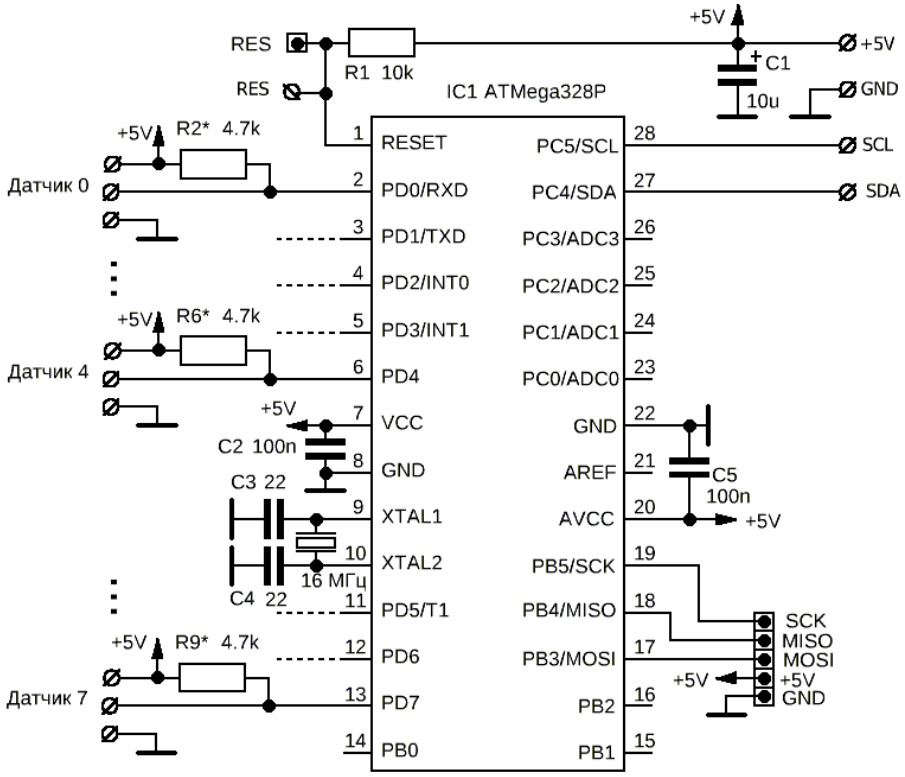

Рис.12. Схема 1-Wire хаба.

Схема 1-Wire хаба показана на рис. 12. В данной реализации к каждому хабу может быть подключено до 8 устройств 1-Wire. Причем, не только всеми любимые DS18B20. Во-первых, абсолютно безо всяких модификаций и дополнений, к нему можно подключать любые "релейные" датчики (концевики, термоконтакты и т. п.). Более того, при реализации схемного «И», к каждому каналу (1-Wire шине) хаба может быть подключено произвольное количество таких релейных датчиков, работающих на замыкание. А, во-вторых, используя АЦП с

интерфейсом 1-Wire (выполненные, например, на универсальных микроконтроллерах, либо на специализированных чипах, типа DS2450 и т.п.), мы можем подключать к 1-Wire хабу любые аналоговые датчики (например, аналоговые температурные датчики, датчики давления типа MPX и т.п.). См. ссылки в разделе 2.6. Другие устройства.

Вариант разводки в виде lay-файла (файл 12_dm_18.12.28.1_pbc_07.lay6) и рисунок платы (файл 12_dm_18.12.28.1_pbc_07.gif) находятся в папке lays архива приложения. Готовая плата хаба показана на рис.13.

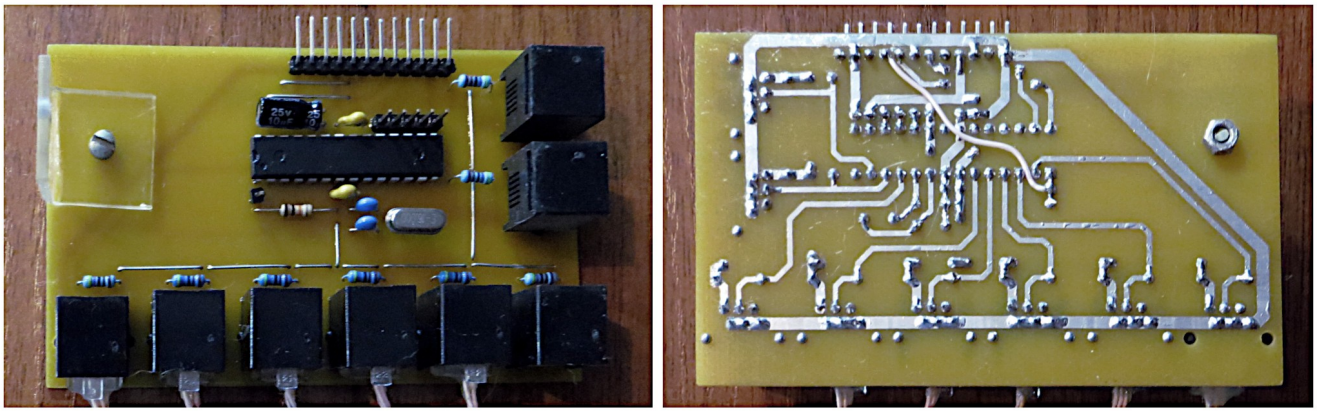

Рис.13. Готовая плата 1-Wire хаба.

Исходный файл прошивки (main.c) и соответствующий hex-файл (main.hex) находятся в папке firmware/A5_Hub архива приложения. Как выполнить прошивку микроконтроллера модуля описано в разделе 2.7. «Прошивка» микроконтроллеров модулей.

Другой, более компактный вариант исполнения 1-Wire хаба можно посмотреть на форуме по ссылке https://forum.homedistiller.ru/msg.php?id=13391417. Фотография этого варианта представлена на рис.14.

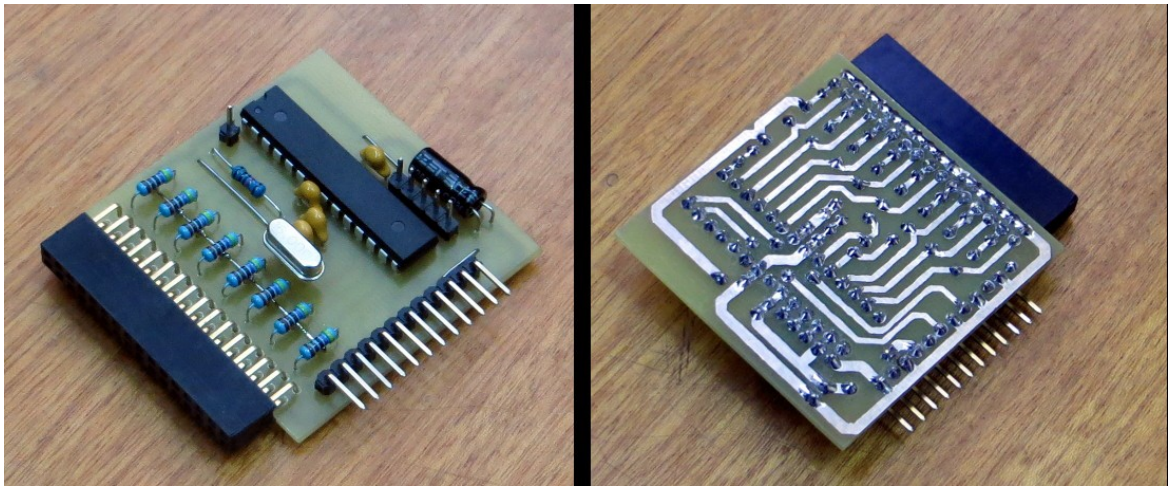

Рис.14. Компактный вариант разводки 1-Wire хаба.

2.6. Другие устройства

Я не стал включать в данное описание ряд устройств, которые были разработаны в рамках данного проекта. Они полезны для разработки расширений автоматики, но не входят в базовый

комплект модулей для автоматизации кубовой ректификации. О них можно прочитать на форуме по ссылкам:

1. 10-разрядный АЦП с 1-Wire интерфейсом — в этом топике: https://forum.homedistiller.ru/msg.php?id=13391417. Если 10-разрядной точности хватает, то это устройство может быть полезно для подключения аналоговых датчиков к 1-Wire хабу.
2. Датчик кубового давления на базе MPX5010DP с 1-Wire интерфесом — в этом топике: https://forum.homedistiller.ru/msg.php?id=13393532.

2.7. «Прошивка» микроконтроллеров модулей

Микроконтроллеры ATMega328P, лежащие в основе каждого модуля, являются программируемыми логическими устройствами. В данном проекте программы для микроконтроллеров (в миру - «прошивки») написаны на языке С. Для того, чтобы микроконтроллер мог исполнять программу, нужно перевести текст программы на языке С в машинные команды и загрузить последовательность этих команд во флэш-память микроконтроллера (память программ). Первой задачей занимаются, так называемые, кросс-компиляторы. Приставка «кросс» означает то, что код для одного процессора (микроконтроллера) компилируется на другом процессоре (компьютере). Второй задачей занимаются программы-загрузчики при помощи специальных устройств - программаторов, которые позволяют физически подключить микроконтроллер к компьютеру.

Существует много вариантов выполнить эти задачи. При помощи различных программных средств, разработанных для различных операционных систем. Поскольку в данной автоматике используется управляющий компьютер Raspberry Pi, работающий под управлением полноценной Unix-подобной операционной системы, то вполне естественно и удобной выполнять операции по подготовке кода и прошивке микроконтроллеров на самой малинке.

Самый простой и удобный способ работы с малинкой — подключить к ней монитор, клавиатуру и мышку. Конечно, тут кому как, но для новичков — это наверняка именно так. Будем считать, что мы это сделали. Как подготовить флэш-карточку с операционной системой Raspbian можно прочитать либо на официальном сайте малинки (https://www.raspberrypi.org/), либо на большом количестве ресурсов в Сети (в том числе русскоязычных). Будем считать что малинка подготовлена к работе.

Для прошивки нужен программатор и программа для прошивки микроконтроллера. Практически в любом радиомагазине (или, например, на aliexpress) можно купить недорогой и очень популярный USBasp программатор. Утилита для прошивки **avrdude** бесплатная. Ее можно скачать и установить непосредственно из Сети командой в терминале:

```
sudo apt install avrdude
```

В архиве к данной документации в папке firmware есть набор папок (по кодам семейств модулей) в каждой из которых находятся по два файла: **main.c** и **main.hex**. Первый из них (**main.c**) это исходный файл прошивки для данного модуля на языке С. Второй (**main.hex**) — это уже откомпилированный и собранный файл прошивки, который можно непосредственно загрузить во флэш-память микроконтроллера при помощи утилиты **avrdude**. Это самый простой способ загрузить готовую прошивку в микроконтроллер, чтобы начать работать с автоматикой.

Здесь мы рассмотрим именно этот способ. Если же нужно что-нибудь изменить в прошивке, то необходимо модифицировать исходный код прошивки (соответствующий файл **main.c**), скомпилировать его, собрать и сформировать новый hex-файл. Как это делать многократно описано на разных ресурсах Сети. Здесь же мы рассмотрим только самый простой способ — использование уже готовой прошивки.

Итак, будем считать, что **avrdude** установлен. Подключаем прошиваемую плату модуля к программатору шестью проводками с соответствующими наконечниками, соблюдая соответствие линий: +5В программатора — к +5В платы, землю к земле|, Reset — к Reset, MISO — к MISO и т. д. Будьте аккуратны! И подключаем программатор к любому порту USB малинки. (рис.15). Можно сделать и отдельный переходник, но, если проводки разноцветные, то можно и просто так, как показано на рисунке.

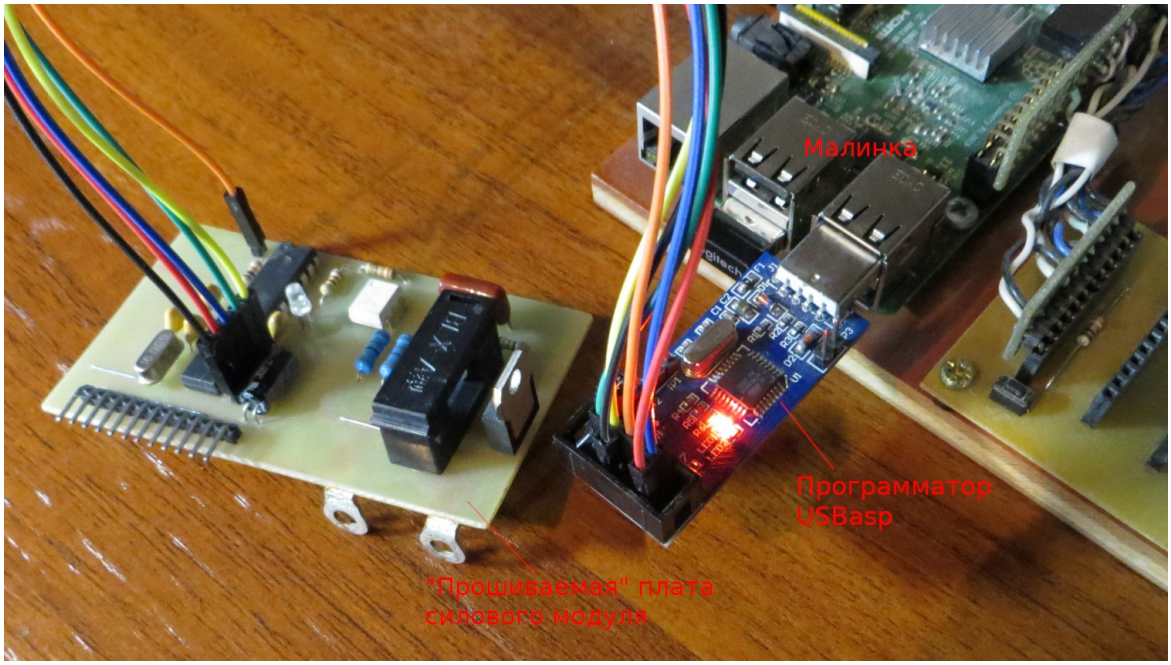

Рис.15. Плата силового модуля и программатор USBasp, подключенные к малинке для прошивки.

Для проверки набираем в терминале команду:

sudo avrdude -c usbasp -p m328p

Если все нормально, **avrdude** считает информацию с микроконтроллера, фьюзы, выдаст эту информацию и поблагодарит. Можно «шить».

Если микроконтроллер новый с завода, то нужно перепрограммировать один байт фьюзов, для того, чтобы тактовая частота микроконтроллера бралась генератора с внешним кварцем 16 МГц, а не от внутреннего генератора, как установлено на заводе. Для этого дадим команду:

sudo avrdude -c usbasp -p m328p -U lfuse:w:0xE7:m

При успешном выполнении этой команды, **avrdude** покажет информацию о фьюзах. Обратите внимание, что содержимое младшего байта фьюзов изменилось на 0xE7.

Ну теперь можно загрузить и саму прошивку. Для этого переходим в терминале в папку с нужной прошивкой (предполагается, что папка firmware из архива к данной документации скопирована в малинку). Например, мы ходим прошить силовой модуль как твердотельное реле. Для этого идем в папку firmware/A2_relay и вводим команду:

sudo avrdude -c usbasp -p m328p -U flash:w:main.hex

avrdude загрузит прошивку во флэш память микроконтроллера, проверит все ли записалось нормально и, конечно же, опять поблагодарит. Все. Отключаем программатор от малинки и модуль от программатора. Микроконтроллер модуля прошит. Модуль готов к установке в крейт и работе.

Так нужно поступить с каждым модулем автоматики.

Внимание! После сборки и прошивки всех модулей необходимо занести в память EEPROM микроконтроллеров некоторых модулей калибровочные данные. К таким модулям, в частности, относятся: датчик RMS и силовые модули с прошивками PDM и PWM, используемые для управления ТЭНами и клапанами отбора. Эту информацию можно записать во время прошивки, но удобнее это сделать средствами библиотеки lite, к рассмотрению которой мы приступаем.

3. Второй уровень — библиотека lite

Снабжение каждого исполнительного устройства и датчика своим микроконтроллером обеспечивает определенный уровень абстракции от рутинных низкоуровневых операций реального времени. Тем не менее, этот уровень еще требует знания и оперирования со многими аппаратурными особенностями системы, такими, как адреса устройств на шине I2C, номера каналов 1-Wire хабов, кодов команд, посылаемых микроконтроллерам и т. д. Классы-обертки, составляющие основу библиотеки lite, позволяют повысить уровень абстракции и работать с устройствами автоматики на уровне физических величин, связанных с соответствующими устройствами. Таких как температура, давление, мощность нагрева, скорость отбора и т. д.

Библиотека lite написана на языке Python, версии 3.x и составляет часть программного обеспечения 2-го уровня, выполняющегося на малинке. Сама библиотека состоит из двух файлов (модулей).

Первый файл **lite.py** содержит классы-обертки для всех типов устройств, описанных в предыдущей главе, и некоторых популярных датчиков, таких как датчики температуры DS18B20 и датчики атмосферного давления BMP180, BMP280, MS5611. Кроме классов-оберток, модуль **lite** содержит ряд сервисных функций для агрегирования информации о текущем состоянии системы, а также функционал для автоматического тестирования, идентификации подключенных устройств, создания и именования соответствующих объектов классов-оберток. Модуль **lite** может загружаться отдельно и использоваться для «ручного» управления установкой непосредственно из консоли Python, тестирования отдельных элементов системы и выполнения сервисных функций. Например, для записи тех же калибровочных данных.

Второй файл библиотеки (**main.py**) содержит средства для организации главного цикла приложений, подгрузки пользовательских скриптов для выполнения специфических операций и организации интерфейса с пользователем. Другими словами, модуль **main** предназначен для организации автоматических режимов работы установки.

В этой же папке **lite_0413** в архиве-приложении к данной документации находятся два файла — примеры пользовательских скриптов (файлы **sr1.py** и **sr2.py**) и универсальный клиент. Для установки библиотеки достаточно просто скопировать папку **lite_0413** в свою рабочую папку на малинке.

Текущая версия библиотеки 0.4.1.3. Рассмотрим эту версию подробнее.

3.1. Классы-обертки

Все классы-обертки устройств, подключаемых к шине I2C являются наследниками класса I2C, в котором реализован функционал для работы с шиной I2C.

Каждый класс-обертка имеет атрибут **fc**, представляющий код семейства данного устройства. На настоящий момент в библиотеке используются следующие коды семейств:

Код	Устройство	Первая буква имени объекта
0xA1	Датчик RMS	U
0xA2	Твердотельное реле	r
0xA3	PDM-контроллер	w
0xA4	PWM-контроллер	q
0xA5	Хаб 1-Wire	H
0xA6	1-Wire 10 bit ADC	
0xA7	Датчик давления MPX5010DP	P
0xA8	Датчик расхода	Q
0xA9	Сервомашинка MG90S	s
0x06	Отладочный макет	
0x28	Датчик температуры DS18B20	T
0xFF	Нормально разомкнутый контакт	Z
0x00	Нормально замкнутый контакт	Z
0x55	Датчик давления BMP180	P
0x58	Датчик давления BMP280	P
0x59	Датчик давления MS5611	P

В первом столбце таблицы код семейства (1-байтовое шестнадцатеричное число), во втором — наименование устройства, в третьем столбце — первая буква имени объекта, соответствующему конкретному устройству. Процедура именования объектов будет подробнее рассмотрена ниже в разделе 3.2. Идентификация устройств и генерация объектов.

Каждый класс-обертка имеет свойство **calibr**. Это список трех чисел. Первое число — коэффициент для перевода кода в значение физической величины, соответствующей данному устройству. Код для датчиков — это обычно код АЦП, а для исполнительных устройств — это обычно код, посылаемый контроллеру устройства. Второе число — это смещение кода. Смещение вычитается из кода перед умножением на коэффициент. Третье число — номер единицы измерения соответствующей физической величины в списке единиц измерения, используемых в библиотеке:

units = ['', 'В', 'Вт', 'мл/час', '°C', 'мм.рт.ст.', 'мл', 'сек', 'мин', 'час', '°', 'deg'].

Свойство **calibr** может быть прочитано (для всех устройств) и изменено (для устройств, где это допустимо).

Каждый класс-обертка имеет свойство **v**. Это — значение физической величины, соответствующее данному устройству. Например, для датчика температуры — это значение его температуры, для контроллера ТЭНа — это мощность его нагрева, для контроллера клапана отбора — это скорость отбора и т. д. Единица измерения задается третьим параметром списка калибровочных данных.

Для большинства датчиков, свойство **calibr** только для чтения. Но для некоторых датчиков (например, виртуальный датчик измерения расхода) это свойство доступно и для записи. Таким путем осуществляется инициализация, сброс датчика.

Некоторые датчики имеют достаточно длительный период измерения, накопления и усреднения данных. Например, 1 сек для датчика RMS, 0.75 сек для датчика DS18B20. Слишком частое обращение к свойству **v** у таких датчиков может привести к ошибкам или блокировке системы. Для таких датчиков предусмотрен некий период «актуальности» данных. По умолчанию это 5 сек, но он может быть изменен путем изменения атрибута **cudtmax**. При первом обращении к свойству **v** происходит автоматический запуск цикла преобразования датчика. Если следующее обращение происходит раньше, чем истек период времени **cudtmax**, то возвращаются данные, полученные в предыдущем цикле преобразования. Если обращение к свойству **v** происходит после истечения периода **cudtmax**, то запускается новый цикл преобразования.

Цикл преобразования таких датчиков может быть инициирован в любой момент последовательным вызовом метода **conv()** и **update()**. Первый метод запускает цикл преобразования датчика, второй — читает данные с датчика и обновляет свойство **v** объекта датчика на малинке. Если датчик подключен к 1-Wire хабу, то методы **conv()** и **update()** необходимо вызывать для объекта соответствующего хаба. В этом случае запускается цикл преобразования для всех датчиков, подключенных к хабу.

Кроме описанных выше, классы-обертки содержат дополнительные методы, атрибуты и свойства, которые могут быть полезны с некоторых ситуациях. Их можно посмотреть в декларациях этих классов непосредственно в исходном коде библиотеки (файл **lite.py**), который подробно прокомментирован.

3.2. Идентификация устройств и генерация объектов

При импорте библиотеки происходит автоматическая идентификация подключенных устройств и генерация объектов соответствующих классов. Это происходит следующим образом. Последовательно, в цикле по всем возможным адресам шины I2C, делается попытка считать код семейства устройства по данному адресу. В случае успешной попытки чтения анализируется считанный код семейства или сам адрес, т. к. некоторые устройства имеют фиксированные адреса на шине I2C. Код семейства или адрес позволяет однозначно идентифицировать тип устройства и создать объект соответствующего класса. Если объектом является 1-Wire хаб, то тут же последовательно просматриваются все 8 каналов данного хаба и делаются попытки чтения их кодов семейств и соответствующая идентификация датчиков.

При создании объектов формируется уникальный hard-ключ устройства, представляющий собой строку по следующему шаблону: **aa_n_cc**, где **aa** — шестнадцатеричный I2C-адрес устройства,

n — номер канала 1-Wire хаба, если устройство подключено к хабу и **cc** — шестнадцатеричный код семейства устройства. Созданный объект заносится в словарь **obj**. Ключами в данном словаре являются сформированные уникальные hard-ключи объектов.

После идентификации всех подключенных устройств производится именование созданных объектов устройств. Имя объекта формируется следующим образом. Первая буква имени определяется типом устройства (см. таблицу в разделе 3.1. Классы-обертки). Далее идет число, представляющее собой номер под которым этот объект был обнаружен в процессе идентификации (среди однотипных устройств). Сформированное имя присваивается соответствующему объекту из словаря **obj**. Параллельно, для удобства, создается еще один словарь объектов **obn**, в котором ключами являются имена этих объектов.

3.3. Другие классы и объекты

Помимо классов-оберток и словарей объектов **obj** и **obn**, библиотека **lite** содержит еще несколько классов и объектов.

Класс Mode. Объекты класса **Mode** представляют режимы работы установки, определяемые состоянием исполнительных устройств системы. Класс **Mode** содержит атрибут **name** (наименование режима) и два метода **init()** и **cond()**. Первый метод вызывается при включении данного режима. В нем происходит инициализация рабочих переменных и настройка исполнительных устройств (мощность нагрева, скорость отбора и т. д.). Во втором методе производится проверка условий для переключения в другой режим или для выполнения каких-либо других действий. Этот метод может вызываться, например, периодически в цикле при проведении процесса.

Список modes. Все доступные режимы работы (объекты класса **Mode**) находятся в списке **modes**. Изначально в этом списке есть только один объект класса **Mode**, задающий режим мониторинга. Остальные объекты класса **Mode** должен создавать и помещать в список **modes** сам пользователь.

Класс Cont. Это вспомогательный класс-контейнер, единственный метод которого возвращает список переменных в контейнере и их значения. Например, в данной библиотеке объект **u** этого класса используется для хранения пользовательских переменных.

Словарь hot_keys. Это дополняемый пользователем словарь «горячих» клавиш, которые могут быть использованы для управления установкой. Пример использования горячих клавишей есть в модуле **main.py** (см. раздел 3.6. Модуль main).

Список uhide. Для сложных процессов количество пользовательских переменных может быть значительным. Многие из них не требуют автоматического вывода на монитор в процессе ректификации. Для таких «скрытых» пользовательских переменных есть специальный список **uhide**. Поместив имя пользовательской переменной в этот список, пользователь может скрыть ее от автоматического показа в терминале.

3.4. Сервисные функции

Библиотека **lite** содержит несколько удобных функций для агрегации информации о системе или ее состоянии. Рассмотрим их подробнее.

Функция hinfo(). Эта функция возвращает информацию обо всем обнаруженном оборудовании в виде одной символьной строки с символами возврата каретки и перевода строки. Каждая подстрока содержит информацию об устройстве в следующей форме: 1) имя объекта python, ассоциированного с данным устройство, 2) hard-код устройства, 3) наименование устройства и 4) калибровочные данные данного устройства. Пример работы функции представлен на рис.16.

Функция minfo(). Функция возвращает информацию о доступных режимах работы в виде одной строки с символами возврата каретки и перевода строки. Каждая подстрока содержит номер режима и его название.

Функция uinfo(). Функция возвращает значения пользовательских переменных в виде одной строки символов с символами возврата каретки и перевода строки. Каждая подстрока содержит имя пользовательской переменной и ее значение.

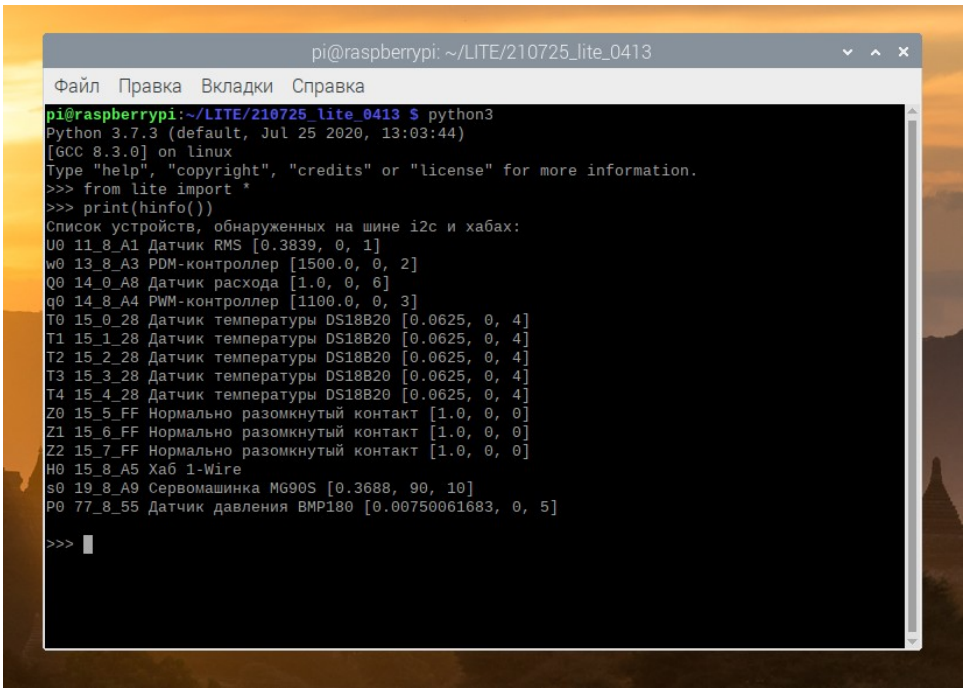

Рис.16. Пример использования функции hinfo().

Функция vinfo(). Функция возвращает текущие значения всех датчиков и контроллеров в виде единой символьной строки с символами возврата каретки и перевода строки. Каждая подстрока содержит имя объекта python, соответствующего устройству и его значение. Пример использования функции **vinfo()** показан на рис.17.

Функция kinfo(). Функция возвращает информацию о горячих клавишах в виде одной символьной строки с символами возврата каретки и перевода строки. Каждая подстрока содержит символ горячей клавиши и ее действие.

Функция tinfo(). Функция возвращает длительность всего процесса и текущего режима в виде единой строки с символами перевода строки и возврата каретки. Пример использования этой функции показан на рис.17.

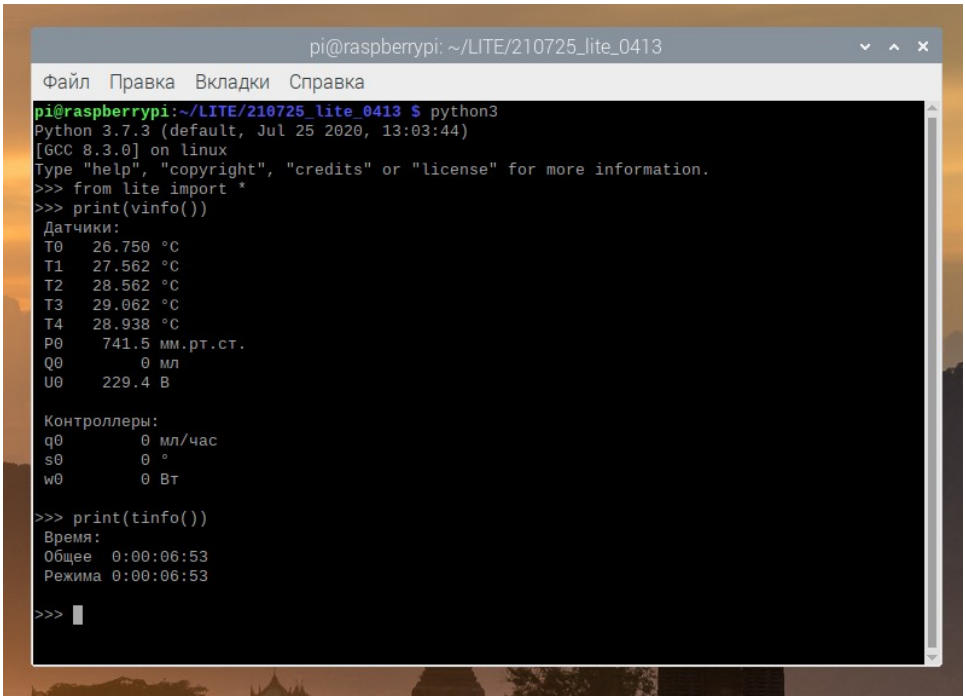

Рис.17. Пример использования функции vinfo() и tinfo().

Функция info(). Функция возвращает всю доступную информация о состоянии системы на текущий момент в виде одной символьной строки с символами возврата каретки и перевода строки. Информация отформатирована для удобного и компактного представления в окне терминала. Пример работы функции info() в рамках скрипта первой ректификации спирта-сырца показан на рис.18.

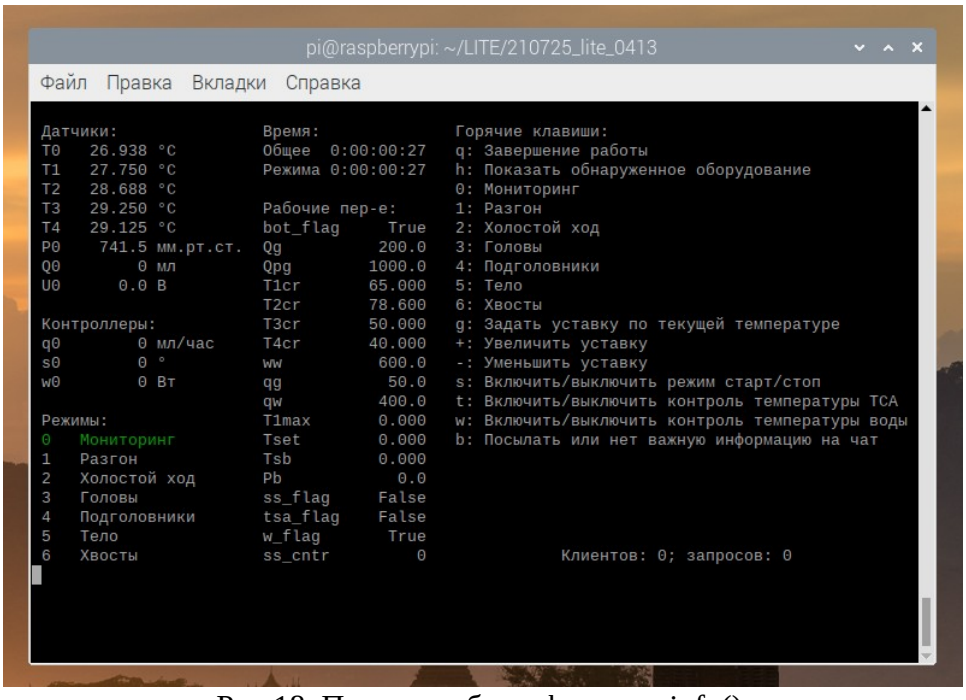

Рис.18. Пример работы функции info().

В библиотеке есть еще пара удобных функций для коллективного запуска циклов преобразований и обновлений данных всех датчиков:

sens_conv() - запуск преобразований всех датчиков,
sens_update() - обновление показаний всех датчиков.

Ну и, наконец, если есть токен telegram-бота и ID чата, то при помощи функции **tele_send_msg()** можно отправить сообщение о состоянии системы в telegram чат.

3.5. Примеры «ручной» работы с библиотекой

В предыдущем разделе уже приводилось несколько примеров работы с библиотекой lite в ручном режиме. Рассмотрим еще пару примеров «ручной» работы с библиотекой из интерактивной консоли python.

3.5.1. Калибровка

Все устройства, используемые в данной автоматике, цифровые. Поэтому их состояние (измеренное значение в случае датчиков или состояние в случае исполнительных устройств) определяется числами (кодами). Для связи кодов и реальных физических величин служат калибровочные данные. Эти данные хранятся в энергонезависимой памяти микроконтроллера EEPROM и могут быть считаны или модифицированы в любой момент. Как это делается при помощи библиотеки **lite** показано на следующем рис.19. Комментарии к командам представлены непосредственно на скриншоте.

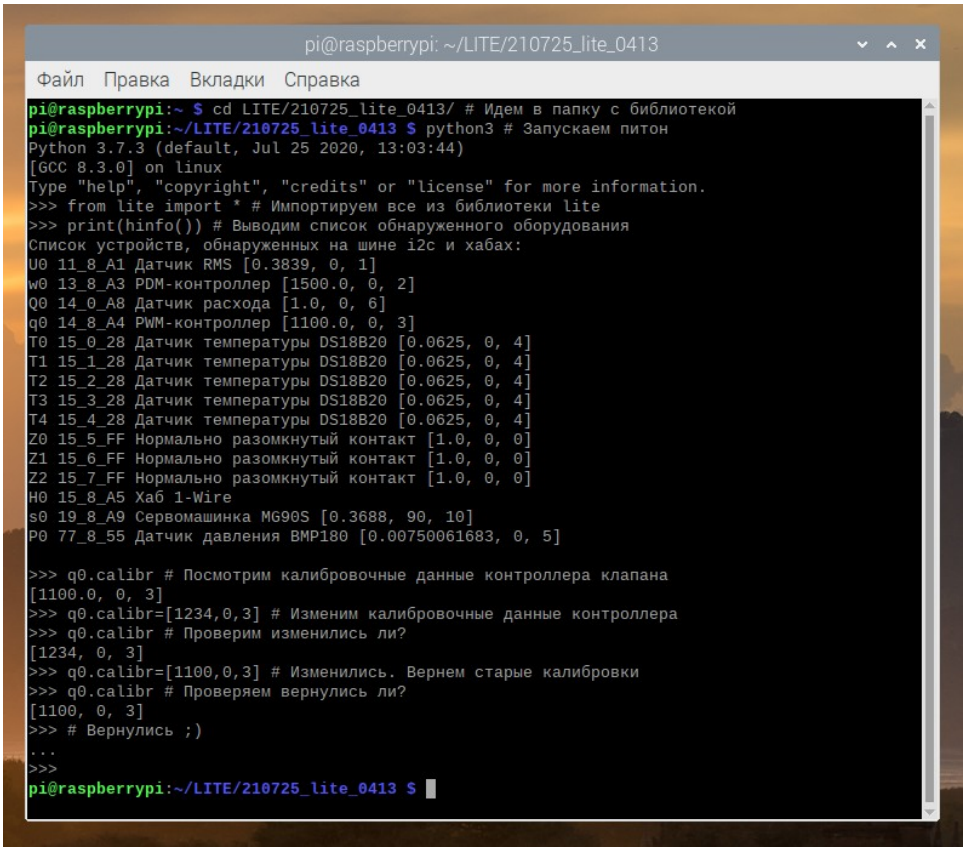

Рис.19. Пример работы с калибровками.

3.5.2. Русские имена объектов

Современный python позволяет использовать имена объектов на русском языке (в unicode). Для этого достаточно просто присвоить новые имена объектам. Пример использования имен на русском языке показан на рис.20.

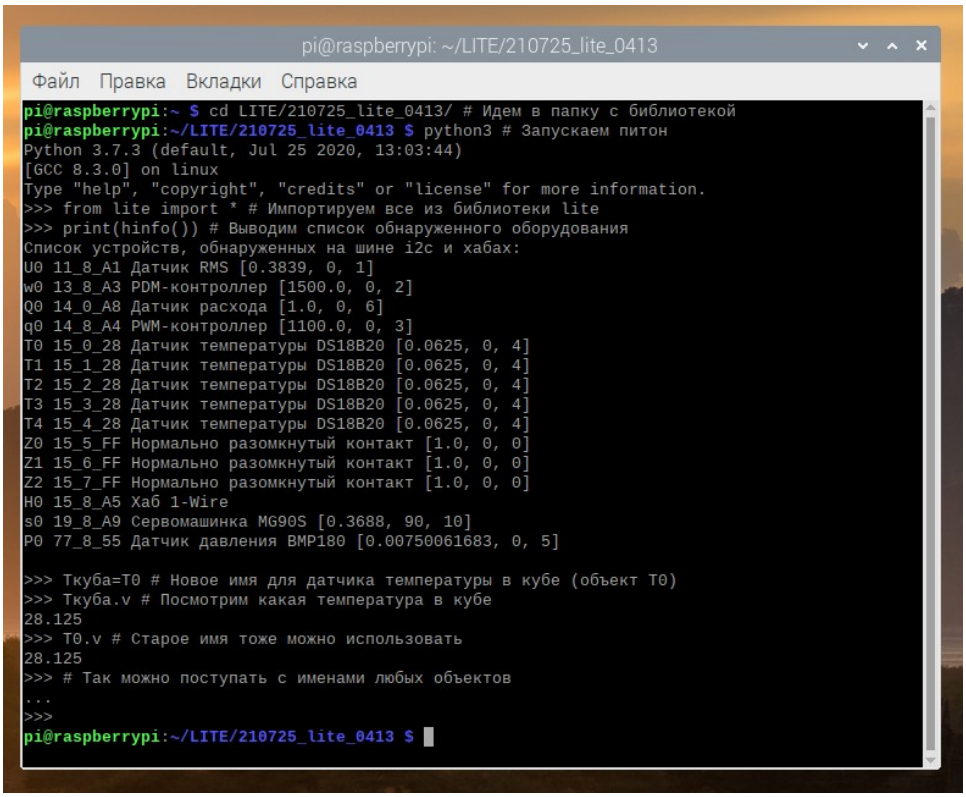

Рис.20. Пример использования русских имен объектов.

3.6. Модуль main

Модуль **main** (файл **main.py**) предназначен для организации автоматических процессов. В этом случае следует запускать модуль **main**, как приложение, командой:

python3 main.py user.py,

где **user.py** — пользовательский скрипт, в котором описаны режимы, переменные и функции, специфические для проводимого процесса. Примеры таких скриптов мы рассмотрим ниже в главе 4. Примеры пользовательских скриптов.

Функционал модуля достаточно прост и его исходный файл **main.py** достаточно подробно прокомментирован. Поэтому ограничимся общим описанием модуля.

В самом начале производится подгрузка и выполнение пользовательского скрипта. Имя пользовательского скрипта задается в командной строке, как было показано выше. По умолчанию делается попытка подгрузить модуль с именем **sr1.py**.

В следующем фрагменте кода производится запуск TCP-сервера в отдельном потоке. IP-адрес хоста (в данном случае — малинки) должен быть указан в пользовательском скрипте в следующей форме:

server_addr = ('xxx.xxx.xxx.xxx', 5238),

где xxx.xxx.xxx.xxx — IP-адрес малинки, 5238 — номер порта, который слушает TCP-сервер. Если кортеж **server_addr** в пользовательском скрипте не определен, то сервер запускается с адресом ('localhost', 5238) для работы клиента непосредственно на хосте (малинке).

Возможно подключение произвольного количества клиентов к TCP-серверу. Клиент посылает серверу строку, которая интерпретируется как горячая клавиша или команда python в пространстве имен работающего приложения. Это дает возможность контролировать текущее состояние системы, изменять режим работы, «на лету» изменять значения любых пользовательских переменных и многое-многое другое. Подробнее клиентское приложение будет рассмотрено в разделе 5. Простой универсальный клиент.

Затем в модуле **main** производится объявление и инициализация рабочих переменных, инициализация таймеров и перевод консоли в неблокирующий режим без эха. В этом случае все нажатия на клавиши обрабатываются непосредственно приложением.

В импортируемом пользовательском скрипте пользователь может определить callback-функцию **crit_info()**, которая будет вызываться при важных событиях. У функции один параметр — текст сообщения. Эта функция может, например, посылать это сообщения в телеграм-чат, на e-mail и т. п. Первый раз эта функция вызывается как раз после завершения всех подготовительных операций и перед входом в главный цикл приложения.

Далее начинается безусловный главный цикл приложения.

В самом начале тела цикла дружно инициируется запуск преобразователей всех датчиков системы. Пока происходит преобразования, проверятся было ли нажатие какой-нибудь клавиши клавиатуры. Если событие имело место — соответствующий запрос помещается первым в очередь запросов. В эту же очередь, в других потоках, помещаются и запросы от клиентов, подключенных к TCP-серверу.

Далее из очереди запросов извлекается первый запрос и актуализируются (обновляются) данные со всех датчиков.

Следующий шаг - анализ извлеченного из очереди запроса.

Если запрос представляет собой нажатие горячей клавиши, то производятся соответствующее этой клавише действие. В конце этого блока условий, вызывается функция **user_hk()**, в которой дополнительно анализируются пользовательские горячие клавиши и совершаются, если необходимо, соответствующие действия.

Если же запросом является сообщение, то оно интерпретируется и выполняется как выражение на языке python.

Следующий блок тела цикла занимается проверкой условий смены режима работы установки и выполнение соответствующих действий.

Далее следует форматирование и вывод данных на консоль и в лог-файл.

Ну и, наконец, в последнем блоке тела цикла производится проверка условий выхода из главного цикла приложения и, если нужно, отправка сообщений клиентам.

Рассмотренный запрос удаляется из очереди и цикл повторяется.

После выхода из главного цикла система переводится в режим мониторинга (0), подключенным клиентам рассылаются сообщения о завершении работы, консоль переводится в нормальный режим работы и закрывается лог-файл.

4. Примеры пользовательских скриптов

Итак, выше мы рассмотрели два варианта работы с библиотекой lite. Первый режим предназначен для «ручной» работы из терминала интерпретатора python. В этом случае, после загрузки интерпретатора, мы импортируем часть библиотеки (модуль **lite**) питоновской инструкцией:

*from lite import **

Во время импорта библиотеки производится автоматический анализ подключенного оборудования и создание всей инфраструктуры, необходимой для полного контроля над оборудование установки. В том числе и возможность работы с автоматически сгенерированными питоновскими объектами, соответствующими всем обнаруженным устройствам. Примеры такой работы были рассмотрены выше в разделах 3.4. Сервисные функции и 3.5. Примеры «ручной» работы с библиотекой.

Второй режим предназначен для организации автоматических процессов. В этом случае производится запуск другой части библиотеки (модуль **main**) как питоновское приложение консольной командой:

python3 main.py user.py,

где **user.py** — пользовательский скрипт, в котором описаны специфические для проводимого процесса детали. Имя и структура пользовательского скрипта может быть произвольным. Он может содержать любые инструкции на языке python, определять пользовательские переменные, пользовательские функции и т. д. Единственная обязательная инструкция — импорт второй части библиотеки в самом начале пользовательского скрипта: **from lite import ***. Тем не менее, для того чтобы воспользоваться всеми возможностями библиотеки, пользовательский скрипт должен включать в себя следующие блоки.

1. **IP-адрес TCP-сервера**. Это — IP-адрес малинки. Этот адрес необходимо указать в пользовательском скрипте если предполагается удаленная работа клиентских приложений для дистанционного контроля и управления процессом. Если адрес не указан — сервер запускается в режиме локальной работы на хосте (малинке).

2. **Токен и ID telegram чата**. Эту информацию необходимо указать в пользовательской скрипте, если предполагается получать некоторые полезные сообщения от работающей установки через telegram-канал. К таким сообщения относятся начало и конец работы установки, переключение режимов работы с текущими параметрами процесса и аварийные ситуации. Если токен и ID чата не указаны, сообщения, естественно, никуда не отправляются.
3. **Список пользовательских переменных с указанием скрытых переменных**. Пользовательские переменные необходимы для определения специфических параметров проводимых процессов и их хранения. Их значения автоматически выводятся на консоль во время работы приложения. Пользовательских переменных бывает много и не все они нуждаются в непрерывном мониторинге. Поэтому пользователь имеет возможность указать, что переменная скрыта и она не будет выводиться на экран.
4. **Режимы работы установки**. По сути, любой процесс ректификации можно представить как последовательной прохождение установкой неких режимов работы (часть параметров которых может изменяться). Режим работы определяется параметрами исполнительных устройств, которые устанавливаются при «включении» данного режима. Каждый режим включает в себя название и две функции. Одна — для инициализации режима, вторая — для проверки условий выхода их данного режима для переключения на другой режим работы. Эти параметры, естественно должен задать пользователь в своем пользовательском скрипте.
5. **Функция emergency()**. Если пользователь планирует контролировать аварийные ситуации, то в пользовательском необходимо определить callback-функцию emergency(), в которой описаны аварийные ситуации, которая система будет автоматически отслеживать.
6. **Пользовательские горячие клавиши**. Если пользователь планирует расширить список системных горячих клавиш, то он может дополнить системный словарь горячих клавиш **hot_keys** своими специфическими клавишами. Для отработки событий, связанных с пользовательскими горячими клавишами, пользователь должен определить функцию **user_hk()**, в которой необходимо описать реакцию системы на их нажатие.

В архиве к данной статье есть два примера пользовательских скриптов, которые подробно прокомментированы и могут быть использованы как прототипы для создания собственных пользовательских скриптов.

4.1. Первая ректификации спирта-сырца

Файл этого пользовательского скрипта (sr1.py) очень подробно прокомментирован, поэтому за деталями реализации лучше обратиться непосредственно к файлу. Остановимся только на общем описании алгоритма ректификации, который реализован в этом скрипте. А именно, на режимах работы.

Режим 1 — разгон колонны. Режим так и называется «разгон». Мощность нагрева ТЭНа максимальная, условие перехода на следующий режим — превышение температуры в нижней части колонны некоторого критического значения (пользовательская переменная u.T1cr = 65.0 °).

Режим 2 — режим холостого хода. Мощность нагрева снижается до рабочего уровня, определяемого пользовательской переменной **u.ww**. Отбора нет. Т.е. вся флегма, сконденсировавшаяся в дефлегматоре возвращается в колонну и стекает вниз. Если систему оставить в таком состоянии, то, рано или поздно, она придет к некоторому стационарному состоянию с установившимся профилем температуры в колонне. В принципе, после установления стационарного состояния можно включать отбор голов. Выход колонны на

стационар обычно происходит минут за 20-30. Поэтому в данном скрипте используется более простой критерий перехода в следующий режим — по времени (30 мин).

Режим 3 — отбор голов. Скорость отбора голов определяется пользовательской переменной **u.qg**. Мощность нагрева — как и в предыдущем режиме — **u.ww**. Критерием перехода к следующему режиму является объем отобранных голов, задаваемых пользовательской переменной **u.Qg**.

Режим 4 — отбор «подголовников». Мощность нагрева определяется той-же пользовательской переменной **u.ww**, а скорость отбора — пользовательской переменной **u.qw**. Так же, как и в предыдущем режиме, критерием перехода к следующему режиму является отбор заданного объема флегмы (пользовательская переменная **u.Qpg**). При отборе подголовников сначала наблюдается небольшой рост температуры в нижней части колонны (T1). Затем она, как правило, немного понижается. Максимальная температура в нижней части колонны в данном режиме (пользовательская переменная **u.T1max**) используется для оценки уставки для старт-стопа в режиме отбора основной фракции (тела).

Режим 5 — отбор тела. В этом режиме используется релейный (двухпозиционный) регулятор средней скорости отбора, известный как «старт-стоп». Как показывает опыт, оптимальное значение уставки (пользовательская переменная **u.Tset**) на несколько квантов датчика температуры превышает максимальное значение **u.T1max**, определенное во время предыдущего режима работы. В данном скрипте превышение составляет 5 квантов датчика DS18B20 (пользовательская переменная, задающая квант — **u.dT**). При задании уставки фиксируются атмосферное давление (пользовательская переменная **u.Pb**) и температура в нижней части колонны (пользовательская переменная **u.Tsb**). Эти параметры используются для коррекции уставки при изменении атмосферного давления. В данном скрипте выбран один из самых простых критериев окончания режима отбора тела — по количеству срабатываний старт-стопа (пользовательская переменная **u.ss_cntr**). Как показывает опыт, такого простейшего критерия окончания отбора тела вполне достаточно.

Режим 6 — отбор исправимых (оборотных) хвостов. Режим старт/стопа отключается, мощность нагрева и скорость отбора такая же, как и при отборе тела. Критерием выхода из этого режима является рост температуры пара в дефлегматоре по сравнению с азеотропом.

Функция **emergency()**, используемая системой для отслеживания аварийных ситуаций, в данном скрипте очень проста. Используется всего два критерия: предельная температур ТСА, задаваемая пользовательской переменной **u.T3cr** и предельная температура вода охлаждения на выходе из дефлегматора, задаваемая пользовательской переменной **u.T4cr**.

Блок пользовательских горячих клавиш и описание реакции системы на их нажатие просты и в дополнительных комментариях не нуждаются.

4.2. Вторая ректификация спирта-ректификата

Еще один пример пользовательского скрипта находится в файле **sr2.py**. Он предназначен для проведения второй ректификации неразбавленного спирта-ректификата, полученного в результате первой ректификации. Этот скрипт проще, чем скрипт первой ректификации. Файл также подробно прокомментирован. Поэтому какие-то дополнительные комментарии не требуются. Единственное, что следует отметить — в скрипте есть пара режимов, которые не

используются непосредственно в процессе ректификации, но используются при подготовке установки ко второй ректификации - режим пропарки колонны и слив жидкости из устройства отбора. Это примеры вспомогательных режимов, не увязанных в процесс ректификации.

5. Простой универсальный клиент

Если процесс отлажен и сырье более-менее стабильно, то процесс ректификации обычно проходит полностью автоматически и не требует каких-то действий пользователя, кроме старта всего процесса. Практически как стирка в стиральной машине-автомате :))). Если возникает желание посмотреть состояние системы, это в любой момент можно сделать либо непосредственно с экрана малинки, либо удаленно, например, при помощи AnyDesk. Тем не менее, иногда возникает необходимость вмешаться в процесс, не останавливая его. Например, поправить какую-нибудь пользовательскую (или системную) переменную в связи с изменившимися условиями.

Для решения такого рода задач, в библиотеке **lite** предусмотрено универсальное клиентское приложение, написанное на языке python. Универсальный клиент соединяется с TCP-сервером, запущенным на малинке и может отсылать серверу запрос в виде текстовой строки. Если строка совпадает с одним из ключей в словаре горячих клавиш, то она интерпретируется системой как реальное нажатие горячей клавиши на хосте (малинке). В противном случае, строка интерпретируется как выражение на языке python. Таким образом пользователь может выполнить любое допустимое выражение python в пространстве имен выполняющегося скрипта автоматизации.

Запускается клиент следующей консольной командой:

python3 client.py xxx.xxx.xxx.xxx,

где xxx.xxx.xxx.xxx — IP-адрес хоста (малинки). Если TCP-сервер запущен в режиме локальной работы (см. IP-адрес TCP-сервера), то клиент может быть запущен на том же самом хосте, что и малинка. В этом случае указывать IP-адрес не нужно.

На рис. 21 показан пример использования универсального клиента для модификации пользовательской переменной **u.Qg,** которая задает количество голов, которое будет отобрано. Ну, предположим, например, что в процессе отбора голов вас осенило, что голов надо бы отобрать немного побольше. Как это сделать «на лету» при помощи универсального клиента показано на представленном скриншоте. Комментарии там же. Клиент запущен на рабочей станции в этой же локальной сети, что и малинка.

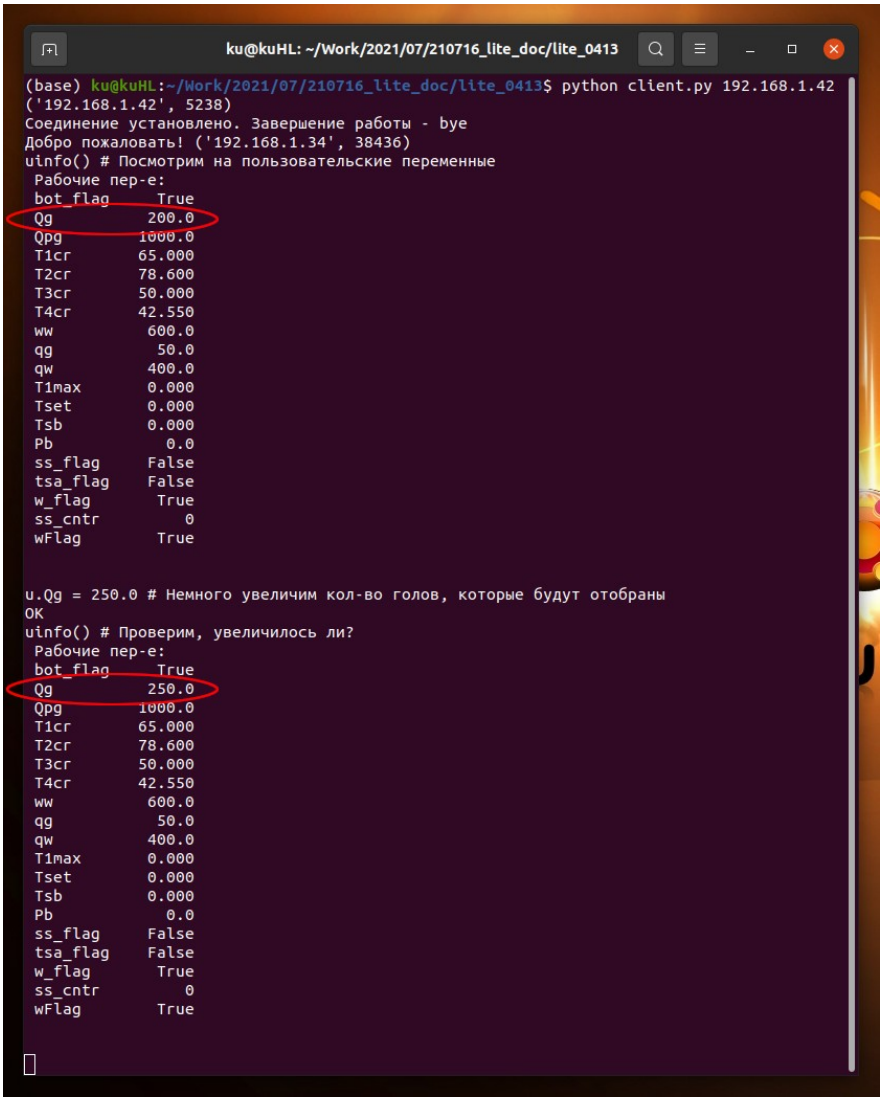

Рис.21. Пример использования универсального клиента для коррекции пользовательской переменной «на лету».