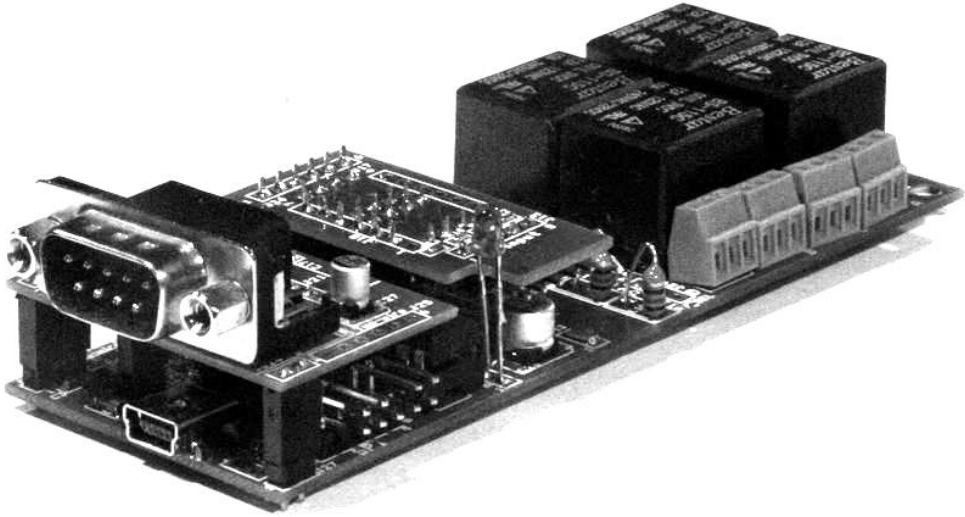
ООО «Юниконтроллерз» Unicontrollers Ltd.

Устройство управления электрическими цепями универсальное четырёхканальное

UNC001

руководство по эксплуатации



Москва – 2014

ООО «Юниконтроллерз»
http://www.unicontrollers.com

Отдел технической поддержки: support.unc@unicontrollers.com

По вопросам заказа и оптовой закупки: order@unicontrollers.com

Содержание

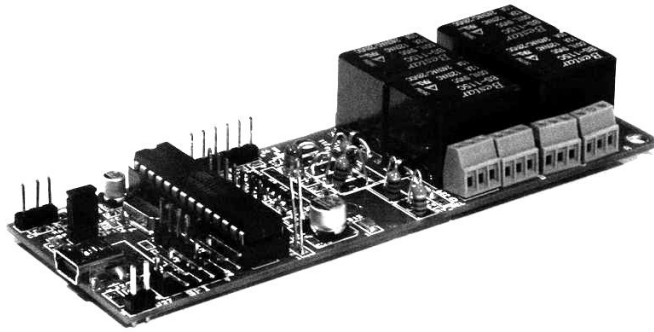


Рис. 1.1: Внешний вид устройства UNC001-1

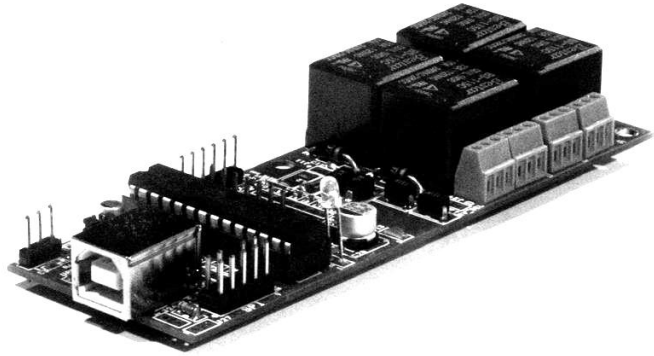


Рис. 1.2: Внешний вид устройства UNC001-2

I. Общее описание

Устройство управления электрическими цепями универсальное четырёхканальное UNC001 (далее «устройство») предназначено для управления электрическими цепями произвольной природы с помощью персонального компьютера. Устройство представляет собой электронную цифровую схему на основе микроконтроллера AVR ATmega8, имеющую порт (USB) для подключения к персональному компьютеру и четыре контактные группы, коммутируемые с помощью электромагнитных реле. Устройство выпускается в четырёх основных модификациях, различающихся по типу электропитания и по виду разъёма USB (см. табл. 1.1).

Все модификации предусматривают возможность подключения дополнительного модуля UNC001/in, который позволяет устройству реагировать на замыкание и размыкание четырёх входных сигнальных линий.

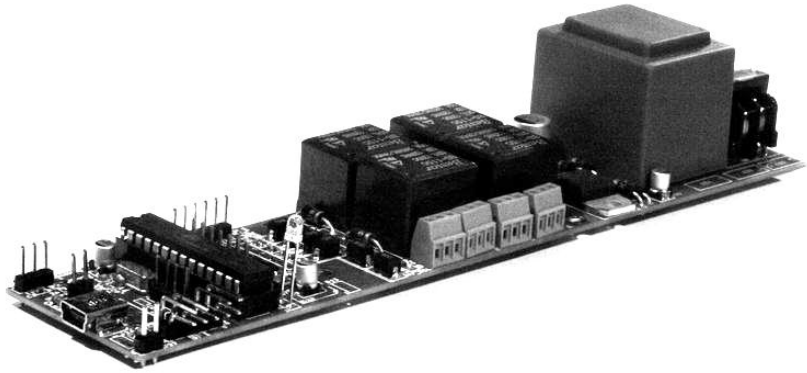


Рис. 1.3: Внешний вид устройства UNC001-3

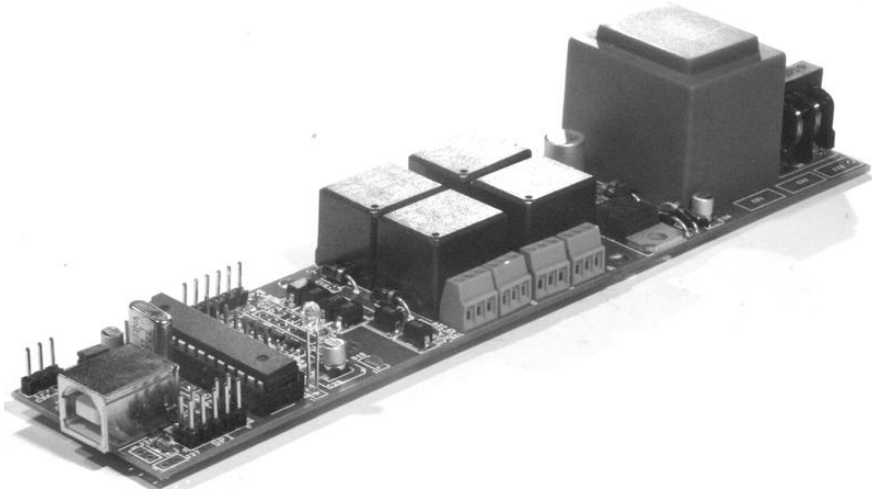


Рис. 1.4: Внешний вид устройства UNC001-4

Доступный ранее модуль UNC001/com в настоящее время снят с производства, поддержка его в программном обеспечении прекращена.

Модификации с питанием от порта USB, являясь полностью низковольтными, допускают эксплуатацию в качестве самостоятельного устройства и при поставке комплектуются шнуром для подключения к персональному компьютеру, оптическим диском, содержащим программное обеспечение и инструкцию по эксплуатации, а также техническим паспортом. Необходимо учитывать, что

Таблица 1.1: Модификации устройства

	питание	разъём
UNC001-1	от порта USB	USB Mini-B
UNC001-2	от порта USB	USB B-type
UNC001-3	от сети 220 В	USB Mini-B
UNC001-4	от сети 220 В	USB B-type

Таблица 1.2: Основные технические характеристики UNC001

Модификация	001-1	001-2	001-3	001-4
Линейные размеры, мм, не более	130x40x20		187x40x35	
Напряжение питания, В	5 пост.		220 В, 50 Гц	
Потребляемая мощность, Вт, не более	1,5		1,3	
Максимальная мощность коммутируемых цепей	300 Вт			

устройства в их исходном виде (без доработки) могут использоваться **только для управления низковольтными цепями** (с напряжением до 40 В).

Модификации с питанием от сети 220 В имеют открытые токоведущие дорожки с напряжением 220 В, что исключает их самостоятельную эксплуатацию; устройство в этой модификации предназначается для встраивания в другие электротехнические устройства и при поставке ничем, кроме технического паспорта, не комплектуется.

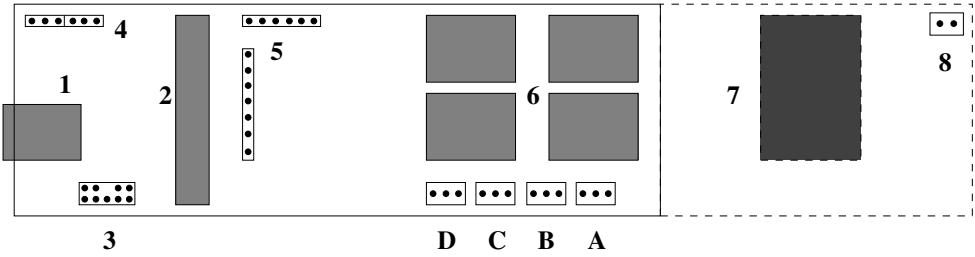
Основные технические характеристики устройств UNC001 приведены в табл. 1.2.

Производитель постоянно работает над совершенствованием устройства. Между вашим экземпляром устройства и настоящим описанием могут существовать различия, не ухудшающие потребительских свойств устройства.

Перед началом работы необходимо внимательно ознакомиться с настоящей инструкцией по эксплуатации.

II. Внешний вид, расположение разъёмов и клеммников

Внешний вид основных модификаций устройства показан на рис. 1.1–1.4. Взаимное расположение основных элементов устройства на монтажной плате схематически изображено на рис. 2.1 (вид сверху; масштаб на схеме не соблюдается). Часть монтажной платы, показанная пунктирной линией и содержащая трансформатор и клеммы для подключения питания от сети 220 В, имеется только в модификациях UNC001-3 и UNC001-4. Для подключения к персональному компьютеру используется разъём **1**. Коммутируемые (выходные) электрические цепи подключаются к клеммникам **A**, **B**, **C** и **D**. Модуль UNC001/in при его наличии устанавливается на контактные группы **5**. Питание (220 В переменного



1. Разъём USB
2. Микроконтроллер AVR
3. Разъём SPI
4. Разъёмы OneWire и TwoWire
5. Место подключения модуля фильтрации входных цепей
6. Электромагнитные реле
7. Трансформатор
8. Клеммы подключения питания 220 В
A,B,C,D. Клеммы коммутируемых цепей

Пунктиром обозначена часть, имеющаяся только в устройствах UNC001-3 и UNC001-4

Рис. 2.1: Схема расположения основных элементов устройства

тока) в модификациях UNC001-3 и UNC001-4 подаётся на клеммник **8**. Разъём SPI (**3**) используется для подключения программатора (в комплект не входит) при смене прошивки устройства.

Контактная группа OneWire (**4**) предназначена для подключения устройств, поддерживающих стандарт OneWire, таких как датчики температуры, электронные ключи–«таблетки» и т. п.

Контактная группа TwoWire в настоящее время не поддерживается прошивкой и не может использоваться.

III. Схемы и способы подключения устройства

3.1. Подключение коммутируемых цепей

Устройство способно управлять коммутацией четырёх независимых электрических цепей, называемых в настоящей инструкции цепями **A**, **B**, **C** и **D**; каждая цепь управляется своим электромагнитным реле. Коммутируемые контакты реле выведены на трёхконтактные клеммники, обозначенные на рис. 2.1 буквами **A**, **B**, **C** и **D**. На рис. 3.1 показана схема подключения клеммника к контактам реле. При выключенной обмотке реле (в том числе и в случае, если электропитание устройства полностью отключено) общий провод (клемма 0) замкнут с клеммой 1, при включённой обмотке реле — с клеммой 2.

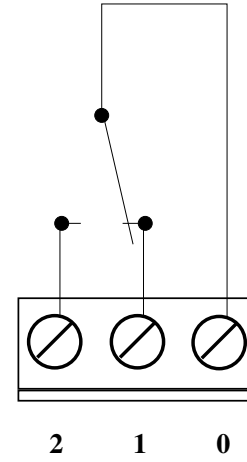


Рис. 3.1: Схема коммутируемой (выходной) цепи

Для подключения к устройству проводов, составляющих коммутируемую цепь, необходимо зачистить концы проводов на 3-4 мм, плоской отвёрткой подходящего размера (например, часовой) ослабить нужные контакты клеммника, повернув соответствующие винты против часовой стрелки на 4-5 оборотов; вставить провода в клеммы; затянуть винты клемм поворотом по часовой стрелке. Обязательно убедитесь, что оголённые части проводов не могут соприкоснуться друг с другом и с окружающими предметами.

3.2. Установка модуля фильтрации входов и подключение входных цепей

Модуль фильтрации входных цепей состоит из двух частей, соединённых между собой шестижильным шлейфом. Первая часть предназначена для установки на устройство UNC001, вторая (клеммник) используется для подклю-

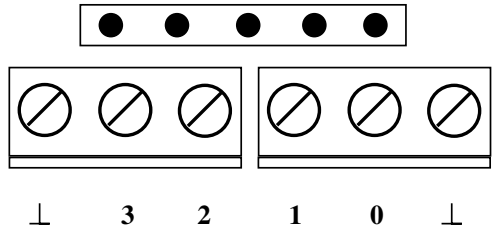


Рис. 3.2: Расположение контактов клеммника для подключения входных цепей

чения входных цепей. Расположенные на основной части модуля контактные группы («гребёнки») представляют собой ответную часть для контактных групп устройства UNC001, обозначенных на рис. 2.1 (см. стр. 7) цифрой **5**. Для установки модуля следует совместить соответствующие контактные группы и плавным движением до упора надеть модуль на «гребёнки» устройства.

Расположение контактов клеммника показано на рис. 3.2. Цифрами **0**, **1**, **2** и **3** обозначены контакты для подключения цепей с соответствующими номерами; знаком «⊥» обозначены два контакта для подключения общего провода; учтите, что эти контакты соединены между собой.

Для подключения проводов, составляющих входные цепи, необходимо зачистить концы проводов на 3-4 мм, плоской отвёрткой подходящего размера (например, часовой) ослабить нужные контакты клеммника, повернув соответствующие винты против часовой стрелки на 4-5 оборотов; вставить провода в клеммы; затянуть винты клемм поворотом по часовой стрелке. Обязательно убедитесь, что оголённые части проводов не могут соприкоснуться друг с другом и с окружающими предметами. Для подключения общего провода можно использовать любой из двух контактов, обозначенных знаком «⊥»; при необходимости можно подключить два проводника к одному контакту.

3.3. Подключение шины 1-Wire®

Шина 1-Wire (в переводе с английского — «один провод») использует один провод для передачи данных и ещё один — в качестве «возвратного» (заземления). Кроме того, возможно использование третьего провода для подачи питания на устройства, подключённые к шине; при отсутствии третьего провода

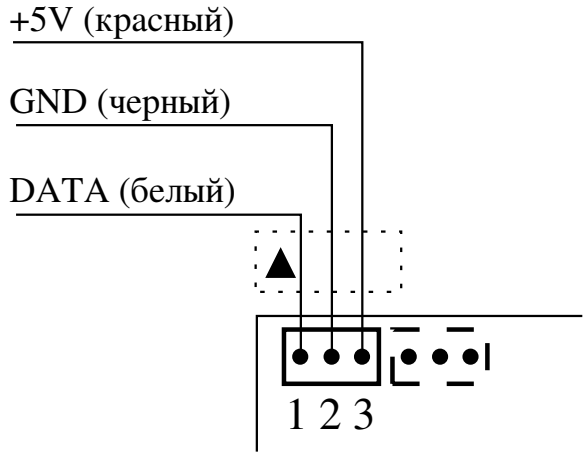


Рис. 3.3: Схема подключения переходника UNC001/1w

устройства, подключённые через 1-Wire, могут получать питание через провод передачи данных (так называемый «паразитный» режим питания).

Устройства UNC001 способны играть роль мастера шины 1-Wire. Для подключения шины к устройству UNC001 используется группа из трёх контактов, расположенная в верхнем левом углу платы (см. рис. 2.1 на стр. 7) и помеченная как *OneWire* или (в зависимости от партии плат) как *1Wire*. Контакты группы пронумерованы слева направо цифрами 1, 2 и 3 и предназначены, соответственно, для подключения провода передачи данных, «возвратного» (заземляющего) провода и дополнительного провода питания.

Компания-изготовитель рекомендует использовать для подключения шины 1-Wire переходник UNC001/1w. В этом случае провод данных, «возвратный» провод и провод питания имеют, соответственно, белый, чёрный и красный цвет; контакт №1 штекера переходника, к которому подключён белый провод, дополнительно помечен значком треугольника (см. рис. 3.3).

Обеспечиваемое напряжение питания — +5 В.

3.4. Подключение питания (только UNC001-3, UNC001-4)

Для подключения питания 220 В переменного тока в соответствующих модификациях устройства используется клеммник **8** (см. рис. 2.1 на стр. 7). Для подключения необходимо зачистить концы проводов на 3-4 мм, при использовании многожильного провода — подкрутить конец провода или облудить его паяльником; плоской отвёрткой подходящего размера (например, часовой) ослабить контакты клеммника, повернув соответствующие винты против часовой стрелки на 4-5 оборотов; вставить провода в клеммы; затянуть винты клемм поворотом по часовой стрелке. Обязательно убедитесь, что оголённые части проводов не могут соприкоснуться друг с другом и с окружающими предметами.

IV. Встроенное программное обеспечение и его возможности

Текст настоящего руководства соответствует версии прошивки 4141014 и более поздним (до выпуска очередной версии руководства). Если ваша версия прошивки имеет меньший номер, используйте версию руководства по эксплуатации, прилагавшуюся к вашему устройству, либо обновите прошивку.

Устройства серий UNC001 и UNC01x управляются встроенной программой «`UNC0xx Firmware`» (далее — «прошивка»), обеспечивающей управление цепями устройства и взаимодействие с управляющим компьютером. Схематическая схожесть устройств UNC001 и UNC01x позволяет использовать в устройствах обоих типов одну и ту же прошивку без изменения.

Производитель постоянно работает над совершенствованием прошивки. Новые версии прошивки доступны на сайте производителя и могут быть загружены

в устройство с использованием программатора UNC501 или другого подходящего программатора; если возможности использовать программатор нет, вы можете заказать у производителя микроконтроллер с установленной в него новой версией прошивки.

С точки зрения операционной системы управляющего компьютера устройства UNC0xx представляют собой HID-устройства и, как следствие, не требуют установки в системе дополнительных драйверов (большинство современных операционных систем уже имеет драйверы для HID-устройств в своём составе). Взаимодействие с UNC0xx через USB позволяет передавать устройству *команды*, а также запрашивать его текущее *состояние*; кроме того, устройство в некоторых случаях выдаёт текстовые *сообщения*, которые в зависимости от настроек могут передаваться через USB в виде *потока ввода*[1]; если поток ввода выключен, устройство сохраняет сообщения во внутреннем буфере, который также можно опросить с управляющего компьютера.

Внутренняя работа устройства происходит *циклами*, причём длительность одного цикла составляет *приблизительно* $\frac{1}{100}$ секунды. Эта длительность является универсальной единицей измерения времени при работе с устройством. Необходимо отметить, что длительность цикла может иметь заметную погрешность. Использовать внутренний таймер устройства для измерения реального времени (например, для включения или выключения цепей в определённое время суток) следует с определённой осторожностью; существенно надёжнее и проще будет использовать для этих целей часы управляющего компьютера, либо, как минимум, регулярно (например, 1 раз в сутки) синхронизировать работу устройства по показаниям часов компьютера.

Прошивка поставляется в соответствии с условиями лицензии GNU GPLv3, с текстом которой можно ознакомиться в сети Интернет, в том числе на сайте производителя устройства. Для устройств, укомплектованных оптическим диском, соответствующий диск содержит полный исходный текст прошивки; также исходные тексты прошивки (включая более новые её версии) можно найти на сайте производителя.

4.1. Параметры настройки и конфигурационные регистры

Пользовательская часть серийного номера позволяет отличать устройства друг от друга при подключении к шине USB одновременно нескольких устройств UNC0xx. Серийный номер устройства состоит из трёх частей, каждая из которых содержит четыре цифры. Первая часть означает номер партии устройств, вторая — номер, идентифицирующий устройство внутри партии, и, наконец, третья часть считается пользовательской и применяется для идентификации устройства при одновременной работе с несколькими устройствами. Исходно эта часть серийного номера для всех приборов установлена в значение 7775, но прошивка позволяет установить любое удобное для вас значение. Если

[1] В настоящее время эта функция доступна только для ОС Linux; под Windows поток USB-ввода не работает.

вы планируете использовать больше одного устройства с одним компьютером, рекомендуется сразу же сменить заводское значение на другое, например `0001`, `0002` и т. д. (см. стр. 24).

Настройки устройства задаются значениями *конфигурационных регистров*.

Регистр аппаратной конфигурации предназначен для настройки прошивки в соответствии с аппаратными особенностями конкретного устройства. Регистр состоит из 32 бит (см. табл. 4.1); старшие 16 бит задают длительность одного цикла в единицах, приблизительно равных $0,6 \times 10^{-6}$ с., и могут быть использованы для тонкой настройки внутреннего таймера; заводское значение составляет $3A98_{16}$ (15000). Младшая половина регистра (биты с 15 по 0) для устройств UNC001-x должна иметь значение `0004`, в противном случае возможна некорректная работа устройства вплоть до его физического повреждения.

Установка слишком низких значений длительности цикла внутреннего таймера может лишить ваше устройство возможности отвечать на запросы управляющего компьютера; восстановить устройство в этом случае можно только с помощью программатора. Если вы не уверены в своих действиях, компания-производитель настоятельно рекомендует воздержаться от экспериментов с регистром аппаратной конфигурации.

Регистр пользовательской конфигурации задаёт наиболее общие параметры работы устройства. Регистр содержит 32 бита; старший байт позволяет задать действия, которые устройство выполнит при включении питания, байт №2 задаёт режим работы с шиной 1-Wire, байт №1 устанавливает режим работы канала вывода, младший байт отвечает за длительность срабатывания входных линий (защита от дребезга). Распределение битов регистра приведено в таблице 4.2. Например, значение `0x00fc0102` включает все имеющиеся функции, связанные с шиной 1-Wire, направляет сообщения в «поток ввода» через USB и устанавливает, что смена статуса входной линии фиксируется по прошествии двух циклов (рекомендованное значение). Значение `0x1a000001`, напротив, выключает все функции 1-Wire, отключает передачу информационных сообщений по USB, смену статуса фиксирует по одному полному циклу, а при старте устройства предписывает выполнить действие, записанное в расширенный регистр `E26` (26 — шестнадцатеричное `1a`). Расширенные регистры подробно рассматриваются ниже.

Таблица 4.1: Регистр аппаратной конфигурации

Биты	маска	назначение
31-16	`0xffff0000`	длительность цикла таймера
15-8	`0x0000ff00`	для устройств UNC001-x должны содержать `00`
7-0	`0x000000ff`	для устройств UNC001-x должны содержать значение `0x04`

Таблица 4.2: Регистр пользовательской конфигурации

Биты	маска	назначение
31,30	0xc0000000	зарезервированы
29-24	0x3f000000	номер E-регистра, содержащего действие для выполнения при старте; 0 – отсутствие действия, 48 – выполнить E00
23	0x00800000	сканировать шину 1-Wire?
22	0x00400000	выдавать в поток ввода идентификаторы подключаемых 1-Wire-устройств?
21	0x00200000	то же для отключаемых устройств
20	0x00100000	Считывать температуру с датчиков DS18x20, а также состояние устройств DS2413?*
19	0x00080000	Сканировать расширенные регистры Exx на предмет идентификаторов «таблеток» iButton?*
18	0x00040000	Сканировать расширенные регистры Exx на предмет событий, связанных с температурой?*
17,16	0x00030000	зарезервированы
15-12	0x0000f000	величина задержки при отключении ведомого устройства 1-Wire
11-9	0x00000e00	зарезервированы
8	0x00000100	генерировать поток информации на USB?
7-0	0x000000ff	антидребезг: количество циклов, после которого входная линия считается сменившей статус

*Некоторые или все из битов 18, 19 и 20 регистра пользовательской конфигурации могут игнорироваться отдельными версиями прошивки с целью экономии программной памяти; в этом случае соответствующие действия выполняются всегда, когда включено сканирование шины 1-Wire (бит 23).

Регистры реакции позволяют устанавливать реакцию устройства на замыкание и размыкание входных (сигнальных) цепей.

С каждой из входных цепей связан свой регистр реакции, содержащий 32 бита. Эти регистры обычно обозначаются R0, R1, ... R7; всего регистров реакции восемь, но устройство UNC001-x позволяет использовать только четыре из них.

В каждом регистре биты с 0 по 7 задают младший байт кода действия при замыкании цепи, биты с 8 по 15 задают младший байт код действия при размыкании цепи. Биты с 16 по 19 задают старшую часть кода действия, общую для случаев замыкания и размыкания. Иначе говоря, итоговый код действия формируется из старшей части, взятой из битов 16–19, и младшей части, взятой из битов 0–7 при замыкании, 8–15 при размыкании. Коды действий подробно рассматриваются на стр. 17.

Бит 29 регистра реакции указывает, нужно ли увеличивать значение соответствующего счётчика при замыкании цепи, бит 30 — при размыкании. Бит 28

Таблица 4.3: Регистр реакции

Биты	маска	назначение
31	0x80000000	зарезервирован
30	0x40000000	увеличивать счётчик при замыкании?
29	0x20000000	увеличивать счётчик при размыкании?
28	0x10000000	сохранять соответствие чётности счётчика состоянию линии?
27	0x08000000	выдавать сообщение о замыкании?
26	0x04000000	выдавать сообщение о размыкании?
25-20	0x03f00000	зарезервированы
19-16	0x000f0000	старшая часть (4 бита) кода действия
15-8	0x0000ff00	младший байт кода действия при размыкании
7-0	0x000000ff	младший байт кода действия при замыкании

принимается во внимание только при одновременно установленных (равных 1) битах 29 и 30; если при этом установлен также и бит 28, то при сбросе счётчика сохраняется соответствие его чётности положению соответствующей цепи (если цепь разомкнута, значение остаётся чётным, если замкнута — нечётным); для этого, если цепь замкнута, при сбросе счётчик не обнуляется, а устанавливается в единицу.

Биты 27 и 26 определяют, будет ли выдаваться сообщение соответственно при замыкании и размыкании линии.

Остальные биты регистра реакций (биты 31, 25–20) зарезервированы для будущих версий; в настоящее время они не используются и должны быть установлены в 0.

Расширенные регистры, обозначаемые E00, E01, ..., E47 (всего 48 регистров), предназначены для создания более сложных реакций на события, а также для хранения информации, связанной с шиной 1-Wire (идентификаторов 1-Wire-устройств, границ температур, считываемых с температурных датчиков, и реакций на пересечение таких границ). Каждый из расширенных регистров состоит из 8 байт, причём старший байт определяет, в какой роли данный регистр используется:

- значения с 0x00 по 0x2f указывают, что регистр содержит идентификатор 1-Wire-ключа (например, «таблетки» iButton), при появлении которого устройство должно исполнить какое-либо действие;
- значение 0x60 означает, что регистр используется для хранения идентификатора ведомого устройства 1-Wire, но никаких автоматических действий, связанных с этим устройством, не предполагается;
- значения 0x70 и 0x7e означают, соответственно, главный и подчинённые регистры для взаимодействия с температурным датчиком;

- значения с `0xe0` по `0xef` обозначают регистр, содержащий список из пяти кодов действий; такие регистры используются при задании сложной логики автономной работы устройства;

- значение `0xf0` указывает, что регистр содержит описание «сложного действия», которое состоит из простого действия (подлежит выполнению немедленно), величины паузы и ещё одного простого действия, которое нужно выполнить по истечении заданной паузы;

- значения с `0xf4` по `0xf7` указывают, что регистр содержит описание «условного действия», выполнение которого зависит от значений в *регистре программного состояния*.

- значения `0x7f` и `0xff` означают, что регистр не используется.

Подробно E-регистры рассматриваются в гл. 6.

4.2. Энергонезависимая память

Значения всех настроек сохраняются в энергонезависимой памяти устройства, при этом все настройки, кроме *расширенных регистров*, при работе копируются в оперативную память. Изменение *пользовательской части серийного номера* и *регистра аппаратной конфигурации* немедленно отражается в энергонезависимой памяти. Изменение *регистра пользовательской конфигурации* и регистров реакции автоматически в энергонезависимой памяти не сохраняются, для этого необходимо дать устройству специальную команду — в противном случае значения этих регистров будут потеряны после выключения питания (отметим, что в некоторых случаях это свойство может быть использовано для проверки экспериментальных настроек). *Расширенные регистры* в оперативную память не копируются, устройство непосредственно использует их значения в энергонезависимой памяти.

Учтите, что ресурс энергонезависимой памяти составляет порядка ста тысяч циклов записи, после чего микроконтроллер вашего устройства будет непригоден к дальнейшей работе. При ручной перенастройке устройства это ограничение, как правило, сложностей не создаёт, поскольку количество перезаписей измеряется десятками, максимум сотнями; тем не менее, имеющееся ограничение необходимо учитывать при написании программ, управляющих устройством — в частности, не прибегать к регулярным автоматическим действиям, предполагающим запись в энергонезависимую память. Например, если написанная вами программа будет изменять какую-то фиксированную часть энергонезависимой памяти один раз в секунду, устройство придёт в негодность приблизительно после полутора месяцев непрерывной работы в таком режиме, в результате потребуется замена микроконтроллера.

Таблица 4.4: Регистр состояния

Биты	маска	назначение
15-12	`0xf000`	зарезервированы
11-8	`0x0f00`	состояние входных цепей
7-4	`0x00f0`	зарезервированы
3-0	`0x000f`	состояние основных выходных цепей

4.3. Состояние устройства

Видимое *состояние* устройства UNC0xx складывается из содержимого двух структур данных (основное состояние и расширенное состояние). В структурах выделяются следующие регистры и области памяти.

Регистр состояния отражает состояние всех цепей, как входных, так и выходных. Этот регистр содержит 16 бит (см. табл. 4.4). Биты с 0 по 3 отражают выходных цепей **A**–**D** (0 — цепь выключена, 1 — цепь включена), биты с 8 по 11 отражают состояние входных цепей (0 — разомкнута, 1 — замкнута). Остальные биты в текущей версии всегда равны нулю.

Кроме того, каждой входной цепи соответствует целочисленный счётчик, способный хранить число от 0 до 65535; каждый из счётчиков можно настроить так, чтобы он увеличивался при каждом замыкании соответствующей цепи, либо при каждом её размыкании, либо при замыкании и размыкании. В последнем случае можно дополнительно скорректировать работу команды обнуления счётчиков так, чтобы она сохраняла чётность счётчика в соответствии с состоянием цепи, то есть чтобы чётные значения счётчика соответствовали состоянию «цепь замкнута», нечётные — состоянию «цепь разомкнута»).

Регистр программного состояния предназначен для задания условных автономных действий и состоит из четырёх подрегистров, называемых **FSM-регистрами**, каждый из которых содержит число от 0 до 15, занимая 4 бита (общий размер регистра программного состояния — 16 бит). Значения FSM-регистров могут изменяться при выполнении специальных кодов действий; в свою очередь, действия, записанные в расширенные регистры с кодами `f4`–`f7`, выполняются в зависимости от текущего значения одного из FSM-регистров, что позволяет описывать нетривиальную логику автономной работы устройства. При старте устройства регистр программного состояния содержит нули. Работа с условными действиями подробно рассмотрена в § 6.3.

Регистр таймера содержит количество *циклов работы*, прошедших с момента включения питания. Цикл приблизительно соответствует 0,01 секунды.

Буфер сообщений представляет собой 32-байтный буфер для хранения символов, организованный по принципу кольцевой записи. Выдаваемые прошивкой сообщения хранятся в этом буфере, причём при его заполнении прошивка продолжает запись сообщений с начала буфера; текущие маркеры начала и конца хранятся в специально выделенных однобайтовых полях. Если *регистром*

Таблица 4.5: Регистр программного состояния (fsm-регистр)

Биты	маска	назначение
15-12	`0xf000`	значение FSM3
11-8	`0x0f00`	значение FSM2
7-4	`0x00f0`	значение FSM1
3-0	`0x000f`	значение FSM0

пользовательской конфигурации задана отправка сообщений через USB в виде «потока ввода», буфер сообщений очищается по мере отправки данных через этот поток. Кроме того, буфер может быть очищен принудительно по команде с компьютера.

Расширенное состояние в нынешней версии прошивки представляет собой структуру данных, предназначенную для информации, полученной при сканировании шины 1-Wire, и включает в себя шесть 8-байтовых регистров для хранения идентификаторов устройств, обнаруженных на шине, а также шесть 2-байтовых регистров для хранения температуры, прочитанной с температурных датчиков; таким образом, количество одновременно подключённых устройств 1-Wire ограничено шестью.

В более старых версиях прошивки (до версии 2131003 включительно) такое ограничение составляло всего четыре устройства, а расширенное состояние содержало, помимо регистров для 1-Wire, также область памяти, предназначенную для некоторых возможностей, которые так и не были реализованы в прошивке. Тем не менее, старый формат расширенного состояния по-прежнему поддерживается хостовым программным обеспечением, что позволяет новым версиям такового работать со старыми экземплярами устройств.

4.4. Коды действий. Управление коммутируемыми цепями

Любые действия с выходными цепями, а также некоторые другие операции задаются с помощью так называемого *кода действия*, который представляет собой 12-битовое беззнаковое целое число. Старшие 4 бита задают код категории действия; интерпретация младшего байта зависит от значения старших битов. Поддерживаемые категории действий перечислены в табл. 4.6.

Старший байт, равный `0`, означает действие с основными коммутируемыми цепями. В этом случае младший байт задаёт *код действия с цепями*. Код действия с цепями представляет собой однобайтовое (восьмибитное) целочисленное значение. Младшие два бита задают действие с цепью **A**, следующие два бита — действие с цепью **B**, 5-й и 6-й биты задают действие с цепью **C**, старшие два бита — с цепью **D**. Комбинация из двух битов задаёт один из четырёх вариантов действия с соответствующей цепью: `00` — не менять состояние цепи (оставить как есть), `01` — перевести цепь в состояние «включено», `10` — перевести в состояние «выключено», `11` — изменить состояние на противоположное. Например, код действия `01010101` (55_{16}) задаёт включение всех четырёх цепей; код `10000000`

Таблица 4.6: Старший полубайт кода действия

код	значение	интерпретация младшего байта
0x00	действие с основными коммутируемыми (выходными) цепями	код действия с цепями
0x01	Для UNC001-х не применяется	
0x02	Для UNC001-х не применяется	
0x09	управление подключённым через шину 1-Wire устройством ввода/вывода на основе DS2413	биты 7–2 — номер расширенного регистра, содержащего 1-Wire-идентификатор соответствующего устройства; два младших бита — новое значение «заслонок» (latches); подробности см. в описании DS2413;
0x0a	(assign) изменение FSM-регистра	старшие 4 бита — номер FSM-регистра (от 0 до 4; бóльшие значения игнорируются); младшие 4 бита — новое значение fsm-регистра
0x0b	(bring) проделать действия, заданные регистрами реакций для заданных входных линий, как если бы линии только что изменили состояние на текущее	битовая строка, задающая, для каких линий это проделать
0x0c	(cancel) отмена «отложенных» действий	0xff — отменить все отложенные действия; 0..47 — отменить отложенное обращение к E-регистру с заданным номером; 0xcn (для n от 0 до 9) — отменить отложенное действие с заданным номером
0x0d	(display) выдача сообщения вида «`A:mm`»	число `mm`
0x0e	(extended) действие, заданное расширенным регистром	номер `nn` регистра `Enn`
0x0f	(fake) пустая операция	игнорируется

(80_{16}) задаёт выключение цепи **D**, остальные цепи оставляет без изменения; код `00001100` ($0C_{16}$) переключает цепь **B** в состояние, противоположное текущему; код `01100101` (65_{16}) выключает цепь **C**, а все остальные, наоборот, включает. Наконец, код `00000000` не делает ничего (все четыре цепи остаются в том состоянии, в котором были).

Код действия используется при подаче команд устройству с управляющего компьютера, а также в *регистрах реакции* для задания действий при замыкании

или размыкании входной цепи и в *расширенных регистрах* для описания более сложных вариантов автономного поведения устройства.

Программное обеспечение, поставляемое с прибором, позволяет задавать *код действия с цепями* одним из следующих способов:

- в двоичной системе — последовательностью восьми цифр 0 или 1;
- в шестнадцатеричной системе с префиксом "x"; например, `xA5` означает включение цепей **A** и **B** с одновременным выключением цепей **C** и **D** (соответствующая двоичная запись — `10100101`);
- в виде группы из четырёх символов, каждый из которых обозначает свой вариант действия: «+» — включение цепи, «-» — выключение цепи, «^» — переключение цепи в противоположное состояние, «=» — отсутствие действия с цепью; например, запись `+-==` означает включение цепи **D**, выключение цепи **C**, а цепи **A** и **B** остаются в прежнем состоянии (соответствующее двоичное значение — `01100000`).

Как видно из таблицы 4.6, 12-битные коды действий могут быть использованы не только для непосредственного управления цепями, но и для других действий; здесь мы приведём их краткое описание, тогда как подробное описание с примерами использования будет дано в разделах настоящего руководства, посвящённых соответствующим возможностям. Обратите внимание, что латинскими буквами `A`–`F` мы обозначаем цифры шестнадцатеричной системы счисления (соответствующие десятичные значения — от 10 до 15).

Коды действий, имеющие старший полубайт `9`, позволяют управлять через шину 1-Wire ведомыми устройствами, основанными на микросхеме DS2413, которые поддерживают два канала ввода/вывода. Идентификатор 1-Wire ведомого устройства необходимо заранее занести в один из выбранных E-регистров, для чего обычно используют код `0x60` (см. § 6). Младший полубайт кода действия формируется из шести бит, задающих номер этого E-регистра, и двух бит, которые следует передать ведомому устройству. Например, при выполнении действия `x9A2` ($A2_{16} = 10100010_2$) будет взят идентификатор 1-Wire, хранящийся в регистре E40 ($40_{10} = 101000_2$), и ведомому устройству с этим идентификатором будут отправлены биты `10`.

Коды действий, имеющие старший полубайт `A`, (мнемонически можно сопоставить этот код со словом *assign*) позволяют изменять fsm-значения в регистре программного состояния, при этом младший байт кода действия используется для задания номера FSM-регистра (биты 5 и 4), а также нового значения FSM-регистра (биты 3, 2, 1 и 0). Например, выполнение действия `xA29` установит регистр FSM2 в значение 9, тогда как команда `xA0B` установит регистр FSM0 в значение B. Подробное описание работы с FSM-регистрами см. в § 6.3.

Коды действий со старшим полубайтом `B` (мнемоническое слово *bring*) предназначены, чтобы для всех или некоторых входных линий можно было выполнить действия, предписанные регистрами реакции (см. § 4.1) к выполнению при

смене состояния, как если бы эти цепи только что перешли в то состояние, в котором в действительности сейчас находятся. Младший байт рассматривается как строка из восьми бит, по одному биту на каждую входную линию; выполнения действий производятся для тех линий, биты которых равны единице. Так, действие `xB01` затронет только первую входную линию и регистр `R0`, действие `xB15` — первую, третью и пятую линии (регистры R0, R2 и R4; двоичное представление числа 15_{16} — 00010101), действие `xBFF` затронет все линии и регистры `Rx`.

Коды действий, начинающиеся с полубайта `E`, (для запоминания можно использовать слово *extended*) позволяют исполнить сложные или условные действия, записанные в E-регистрах; младший байт кода действия задаёт номер E-регистра. Например, код действия `xE1A` заставит устройство выполнить сложное или условное действие, записанное в регистре E26 (`1A` представляет собой шестнадцатеричную запись числа 26). Коды действий, начинающиеся с полубайта `C` (мнемоническое слово *cancel*), предназначены для отмены выполнения «отложенных» действий, при этом младший байт кода может задавать номер регистра, выполнение которого отложено (например, код действия `xC1A` отменит отложенное выполнение регистра E26, если таковое в текущий момент есть); код `xCFF` отменяет все отложенные действия, активные в текущий момент; коды вида `xCCn` (где `n` — цифра от `0` до `9`) отменяют отложенное действие, находящееся в n-ной строке таблицы отложенных действий. Подробное описание возможностей E-регистров вы найдёте в §§ 6.1, 6.2 и 6.3.

Коды действий, начинающиеся с полубайта `D`, (мнемоническое слово *display*) предназначены в основном для отладки прошивки; выполнение такого действия выдаёт шестнадцатеричное представление младшего байта кода в информационный поток.

Полубайт F (мнемоническое слово *fake*) обозначает псевдодействие, выполнение которого на самом деле ничего не делает и не изменяет. Значение младшего байта игнорируется.

V. Программное обеспечение для персонального компьютера

5.1. Состав, инсталляция и сборка

Программное обеспечение для работы с устройствами UNC0xx состоит из следующих основных компонентов:

1. утилита командной строки `uncctl`;
2. утилита `unc_chown` (только в Unix-системах);
3. программа UNC Monitor с графическим интерфейсом пользователя;
4. библиотека управляющих функций `unc0xx` для использования в программах на языках программирования Си и Си++.

Всё программное обеспечение распространяется в соответствии с условиями лицензии GNU GPL v.3 с полным комплектом исходных текстов. Рекомендуется использование операционных систем семейства Unix (в том числе GNU/Linux); хотя версии для систем Microsoft Windows также доступны, некоторые функции в них не работают.

Программа UNC Monitor поставляется с демонстрационными целями и в данном руководстве не рассматривается. Управление всеми функциями устройства можно осуществить с помощью утилиты `uncctl` (см. стр. 23). Библиотека `unc0xx` (см. стр. 45) позволяет пользователю самостоятельно разрабатывать программы для работы с устройством на языках Си и Си++, а также с использованием других языков программирования, для которых система программирования предусматривает вызов подпрограмм, написанных на Си.

Установка программного обеспечения в операционных системах семейства Unix (включая GNU/Linux) допускает два варианта: установка готовых исполняемых файлов (в настоящее время доступно только для Linux на платформе i386) и сборка из исходных текстов.

Для установки готовых исполняемых файлов скачайте их (это файлы `uncctl`, `unc_chown` и `uncmon`) с сайта производителя, либо, при наличии в комплекте устройства оптического диска, возьмите эти файлы из каталога `Linux` на диске и скопируйте их в каталог `/usr/local/bin` в вашей системе (это потребует полномочий администратора — пользователя `root`). Программы сразу же готовы к работе. Исполняемый файл программы `unc_chown` в некоторых случаях целесообразно снабдить битом SUID; подробности см. ниже в §5.3.

Сборка программного обеспечения из исходных текстов требует наличия в системе установленной библиотеки `libusb` (см. `http://www.libusb.org/wiki/libusb-1.0`). В большинстве дистрибутивов ОС GNU/Linux эта библиотека уже есть в виде готового пакета и может быть установлена штатными средствами дистрибутива; так, в дистрибутивах Debian, Ubuntu и некоторых других следует дать команду

```
apt-get install libusb-dev
```

В случае, если в вашем дистрибутиве такого пакета не найдётся (что маловероятно), следует скачать исходные тексты библиотеки с вышеупомянутого сайта и воспользоваться инструкцией по сборке и установке библиотеки. Вам также понадобятся компилятор `gcc` и система сборки GNU Make; скорее всего, они уже установлены в вашей системе.

Исходные тексты программ `uncctl`, `unc_chown`, UNC Monitor, библиотеки `unc0xx` и прошивки находятся в файле `unc0xx-NNNNNNN.tgz`, где `NNNNNNN` — номер версии (например, 4141014); раскройте архив командой

```
tar -xzf unc0xx-NNNNNNN.tgz
```

В текущей директории появится поддиректория `unc0xx-NNNNNNN`, в которой, в свою очередь, находится поддиректория `commandline`, содержащая исходные тексты библиотеки и программы; зайдите в неё:

```
cd unc0xx-NNNNNNN/commandline
```

Для сборки нужно запустить утилиту GNU Make; в операционных системах семейства GNU/Linux эта утилита, как правило, называется просто `make`, в других Unix-системах (FreeBSD, SunOS и т. д.) может потребоваться команда `gmake`. Достаточно запустить эту утилиту без параметров, и всё, что нужно (файлы `uncctl`, `unc_chown` и `libunc0xx.a`), будет собрано. Учтите, что программа `unc_chown` по умолчанию собирается в виде динамического бинарного файла, то есть зависит от наличия в системе файлов динамически подключаемых библиотек. Для сборки статической версии можно воспользоваться командой `make unc_chown_static`, что приведёт к сборке исполняемого файла, аналогичного тому, что поставляется на диске и доступен на сайте производителя (полностью статического). В этой версии программа не поддерживает работу с нелокальными пользователями и группами.

В случае возникновения проблем со сборкой для начала убедитесь, что в вашей системе соблюдены все необходимые условия (библиотека `libusb` установлена и доступна, имеется компилятор `gcc` и система сборки GNU Make). Если эти условия выполнены, а сборка всё равно не проходит, пожалуйста, сообщите об этом разработчикам через контактную форму на сайте `http://www.unicontrollers.com`.

В текущей версии не предусмотрена автоматическая инсталляция; это, скорее всего, будет изменено в ближайшем будущем. Так или иначе, для инсталляции программы `uncctl` рекомендуется скопировать её в директорию `/usr/local/bin` или `/usr/local/sbin`; для инсталляции библиотеки скопируйте `libunc0xx.a` в директорию `/usr/local/lib`, а файл `unc0xx.h` — в директорию `/usr/local/include`.

Установка программного обеспечения в операционных системах семейства Windows сводится к копированию файла `uncctl.exe` (а также, при желании, файла uncmon.exe) в любую директорию, доступную через `PATH` (или любую другую, но тогда придётся для запуска программ каждый раз указывать полный путь). Программа `uncctl.exe` не требует никакой специальной инсталляции, не использует дополнительных файлов, не затрагивает системные настройки и должна штатно работать, будучи запущенной из любого места системы, в том числе и со сменного носителя. Если в вашем случае это оказалось не так, свяжитесь с разработчиками. Программа `uncmon.exe` (утилита с графическим интерфейсом) использует в работе файл конфигурации, записываемый в текущую директорию.

Сборка из исходных текстов под Windows возможна с помощью пакета MinGW. Процесс такой сборки полностью аналогичен описанному выше процессу для Unix, с той разницей, что вместо команды `make` следует использовать команду `make -f Makefile.mingw`. Программа `unc_chown` для Windows не собирается (и не требуется).

5.2. Управление устройством с помощью программы uncctl

Программа `uncctl` (в системах семейства Windows — `uncctl.exe`) представляет собой утилиту командной строки, обеспечивающую доступ ко всем функциям устройств UNC0xx. Будучи запущенной без параметров, программа выдаёт справку по её использованию; справка выдаётся на английском языке. В качестве русскоязычного описания используйте настоящую инструкцию.

Просьба учесть, что при работе с ОС семейства Unix (в том числе GNU/Linux) вам, скорее всего, потребуются полномочия системного администратора (пользователя `root`). Этого можно избежать с помощью утилиты `unc_chown`, которая подробно описана в § 5.3. Программа `uncctl` при работе в ОС Unix выдаёт предупреждение, будучи запущена от имени пользователя, отличного от `root`; это предупреждение можно отменить указанием флажка `-w`.

Команда `uncctl -L` выдаёт список устройств UNC001 и UNC01x, подключённых к данному компьютеру через интерфейс USB. Если ни одного устройства не найдено, выдаётся строка `No devices found`. Учтите, что в ОС семейства Unix (в том числе Linux) это может произойти вследствие того, что программе `uncctl` не хватило полномочий; в этом случае попытайтесь вновь запустить её, уже с правами суперпользователя (например, через команду `sudo`). В системах семейства Windows, как правило, ограничений по доступу к устройствам нет. В выдаваемом списке устройств указывается пользовательская часть серийного номера устройства (идентификатор), а также серийный номер полностью (его третья часть совпадает с идентификатором) и номер версии прошивки; при обнаружении устаревшего устройства, протокол которого не поддерживается текущей версией `uncctl`, программа печатает слово `DEPRECATED`. В конце выдаётся общее количество обнаруженных устройств. Например, результат выполнения команды `uncctl -L` может выглядеть так:

```
<id>  [serial number ]   version
7775  [0002:0027:7775]   3140831
1234  [0002:0043:1234]         0   DEPRECATED
2 UNC001 device(s) found
```

Устройство, помеченное словом `DEPRECATED`, управляться имеющейся версией утилиты `uncctl` не может; в таком устройстве необходимо сменить версию прошивки. Для этого можно воспользоваться любым программатором, совместимым с AVR ByteBlaster, в том числе (но не обязательно) программатором UNC501. Образ свежей прошивки и инструкции по её обновлению можно найти на сайте ООО «Юниконтроллерз».

Все команды, поддерживаемые программой `uncctl`, кроме команды `-L`, выполняют какие-либо действия только с одним устройством. В случае, если к компьютеру подключено больше одного устройства, следует с помощью опции `-i` указать идентификатор того, с которым вы хотите работать. Например, команда

```
uncctl -i 3333 -Q
```

будет работать с устройством, идентификатор которого равен `3333`. Опцию `-i` можно использовать совместно с любой из команд, перечисленных ниже. Когда к компьютеру подключено только одно устройство, указание идентификатора не обязательно. В случае, если устройств подключено несколько, а идентификатор не указан, программа изберёт для работы первое из обнаруженных устройств. Пользоваться этим настоятельно не рекомендуется, поскольку в разных обстоятельствах первым может обнаружиться любое из подключённых устройств; программа в этой ситуации выдаёт предупреждение.

Изменить идентификатор устройства можно командой `uncctl -T`; например, команда

```
uncctl -i 3333 -T 0001
```

найдёт устройство с идентификатором `3333` и сменит его идентификатор на `0001`.

Управление выходными цепями, а также другие действия, задаваемые *кодом действия* (см. § 4.4 на стр. 17) осуществляются командой `uncctl -A`, которая требует указания параметра — кода действия. Код действия может задаваться в двоичной или шестнадцатеричной системе, либо с помощью символических обозначений действия. Например, команды

```
uncctl -A +-^=
uncctl -A 01101100
uncctl -A x6C
```

выполнят одно и то же действие, а именно, цепь **A** будет оставлена в прежнем состоянии (`00`), цепь **B** будет переключена в состояние, противоположное текущему (`11`), цепь **C** будет выключена (`10`), а цепь **D** — включена (`01`). Действия можно задать не для всех цепей; например, команда

```
uncctl -A +
```

включит цепь **A**, а команда

```
uncctl -A +-
```

включит цепь **B** и выключит цепь **A**, не затрагивая остальные цепи. При одновременном использовании нескольких устройств следует использовать опцию `-i` для выбора устройства.

Для задания расширенных кодов действий обычно применяется шестнадцатеричная система. Например, команда

```
uncctl -A xe1a
```

заставит устройство выполнить сложное действие, заданное регистром `E26` (шестнадцатеричное `1a` соответствует десятичному 26); команда

```
uncctl -A xcff
```

предпишет устройству сбросить все «отложенные действия», которые могли появиться при выполнении сценариев, заданных регистрами `Exx`.

Действия со старшим полубайтом `9`, предназначенные для работы с ведомыми устройствами 1-Wire на основе DS2413, также можно задать мнемонически в виде строки, в которой первые два символа соответствуют передаваемым битам (и должны быть `+`, `-` и `.`), затем пишется символ `@`, после которого идёт номер E-регистра в десятичном или шестнадцатеричном виде. Например, `+-@26` или `+-@x1a` соответствуют коду действия `x96a`.

Устройство позволяет выполнить заданное действие не сразу, а после определённой задержки. Для этого в дополнение к флагу `-A` указывается опция `-d` с параметром, задающим размер задержки. По умолчанию размер задержки задаётся в *циклах*, приблизительно соответствующих сотым долям секунды. Вы также можете использовать букву `s` для обозначения секунд, `m` для обозначения минут и `h` для обозначения часов. Например, команда

```
uncctl -A 1:---- -d 45m
```

запланирует выключение всех линий первой дополнительной группы через 45 минут от текущего момента. Устройство хранит отложенные действия в таблице из десяти строк; по умолчанию новое действие размещается в первой свободной строке таблицы. Вы также можете задать опцию `-l`, параметр которой должен представлять собой цифру от `0` до `9`. В этом случае отложенное действие размещается в заданной строке таблицы; если в этой строке уже находилось отложенное действие, оно замещается новым. Например, команда

```
uncctl -A xe21 -d 30s -l 5
```

предпишет устройству выполнить расширенный регистр E33 (21_{16}) через 30 секунд, причём предписание будет размещено в строке № 5, возможно, затерев отложенное действие, которое уже там было.

Для опроса и настройки входных линий можно воспользоваться командами `-Q`, `-Z` и `-R`. Команда `uncctl -Q` выдаёт полную информацию о состоянии устройства, а именно значение регистров статуса, пользовательской и аппаратной конфигуации, значения регистров реакции и всех счётчиков; сообщения, накопившиеся в кольцевом буфере сообщений; при работе с шиной 1-Wire выдаются также идентификаторы устройств, обнаруженных на этой шине, для температурных датчиков — считанная с них температура, для устройств на базе DS2413 — их состояние. При наличии у устройства отложенных действий выдаётся список этих действий с указанием оставшегося времени до каждого из них. По умолчанию значения выдаются в краткой форме; для более подробной информации следует добавить опцию `-v`, например:

```
uncctl -Q -v
```

Команда `uncctl -Z` считывает значения счётчиков, обнуляет все счётчики и выдаёт прочитанные значения, т. е. за одно действие снимает информацию из

счётчиков и обнуляет их. Работа команды `-Z` имеет важную особенность для тех входных линий, на которых счётчики настроены на приращение как при замыкании, так и при размыкании линии. В такой ситуации счётчик должен, как легко видеть, иметь чётное значение, если соответствующая линия разомкнута, и нечётное, если она замкнута. Однако если счётчик обнулить (то есть буквально сделать равным нулю) в момент, когда линия замкнута, значение соответствующего счётчика станет равно 0, то есть станет чётным, что в дальнейшем не позволит использовать чётность значения счётчика для определения состояния линии. Устройство поддерживает режим «сохранения чётности счётчика», который включается установкой 28го бита в соответствующем регистре реакции. Если счётчик настроен на приращение при замыканиях и размыканиях, режим сохранения чётности установлен и линия замкнута, то в счётчике после выполнения команды `-Z` будет значение 1, а не 0, что сохраняет возможность использовать чётность. Необходимо отметить, что значение, показываемое командой `-P`, в любом случае будет равно значению счётчика перед его сбросом (неважно, был ли счётчик сброшен в 0 или в 1), так что при сбросе счётчика в единицу возникает «лишняя» единица; если необходима информация о точном количестве срабатываний счётчика, то при использовании команды `-Z` для линий, чьи счётчики настроены на приращение при замыкании и размыкании и сохранение чётности, следует полученные нечётные значения уменьшать на 1.

Команда `uncctl -R` предназначена для настройки входных линий; точнее говоря, эта команда позволяет установить новые значения для регистров `R0`...`R7` (см. описание регистров на стр. 13). Команда требует указания двух параметров; первый из них должен представлять собой цифру от 0 до 7, задающую номер регистра[1], второй задаёт новое значение регистра. Значение можно указать двумя способами. Первый способ предполагает использование шестнадцатиричного 32-битного целого числа, т. е. восьми шестнадцатиричных цифр; например, команда

```
uncctl -R 2 20000055
```

настроит входную линию № 2 на приращение счётчика только при замыкании, плюс к тому при замыкании линии будет выполняться код действия `x55` (двоичное `01010101`), означающий включение всех четырёх основных линий. Команда

```
uncctl -R 0 40000201
```

настроит входную линию № 0 на приращение счётчика только при размыкании, при этом при замыкании линии будет включаться реле **A** (код действия `x01`, двоичное `00000001`), при размыкании реле **A** будет выключаться (код действия `x02`, двоичное `00000010`). Команда

```
uncctl -R 1 60000000
```

[1] Для устройства UNC001-х смысл имеют только регистры с номерами от 1 до 3.

настроит входную линию № 1 на приращение счётчика как при замыкании, так и при размыкании, без выполнения каких-либо действий с выходными цепями; это соответствует исходным заводским настройкам для всех линий. Режим сохранения чётности можно включить командой

```
uncctl -R 1 70000000
```

Второй способ задания значения регистра предполагает использование символических обозначений в кодах действий с цепями. Старшие два байта регистра при этом по-прежнему задаются шестнадцатеричными цифрами, а байты кодов действий отделяются от старшей части и друг от друга символом двоеточия. Например, вышеприведённые команды можно задать и так:

```
uncctl -R 2 2000:====:++++
uncctl -R 0 4000:===-:===+
uncctl -R 1 6000:====:====
uncctl -R 1 7000:====:====
```

Команда `uncctl -G` позволяет изменить значение регистра пользовательской конфигурации; новое значение задаётся в шестнадцатеричном виде. Например,

```
uncctl -G 0
```

отключит защиту входных линий от «дребезга», а

```
uncctl -G 000000ff
```

установит максимально возможное значение количества проверок (в реальном использовании не рекомендуется, поскольку время реакции устройства на изменение состояния линии составит несколько секунд).

Настройки устройства, установленные командами `-R`, и `-G`, действуют до выключения устройства, если только не записать их в энергонезависимую память (EEPROM). Запись текущих настроек в EEPROM осуществляется командой `uncctl -W`. При необходимости можно восстановить настройки, записанные в EEPROM, с помощью команды `uncctl -E`; текущие настройки при этом теряются. Наконец, можно восстановить в качестве текущих исходные заводские настройки (а именно, значение `60000000` во всех регистрах `R0`...`R7`) с помощью команды `uncctl -F`.

Значение *регистра аппаратной конфигурации* может быть изменено командой `uncctl -H`; первым параметром нужно указать новое значение регистра в шестнадцатеричной системе, вторым — слово «`really`», чтобы подтвердить, что вы **действительно** решили изменить этот регистр. **Компания-производитель настоятельно рекомендует воздержаться от экспериментов с этим регистром и снимает с себя всякую ответственность за возможные последствия**.

Для записи в регистры `E00`–`E47` предназначена команда `uncctl -Y`. Первый параметр — номер регистра (десятичное число от 0 до 47; можно также использовать шестнадцатеричную систему, числа от `x00` до `x2F`), второй параметр —

последовательность из ровно 16 шестнадцатеричных цифр, соответствующих восьми байтам регистра.

```
uncctl -Y 40 f00000ff01010e28
```

занесёт в регистр `E40` указание немедленно переключить все основные выходные цепи в противоположное состояние (`00ff` в 5-4 байтах), а спустя приблизительно одну секунду (`0101` в 3-2 байтах) вновь исполнить содержимое регистра `E40` (`0e28` в двух младших байтах). Если дать теперь команду

```
uncctl -A xe28
```

то устройство будет приблизительно один раз в секунду переключать все основные выходные цепи в противоположное состояние до тех пор, пока вы либо не отмените это командой `uncctl -A xcff` или `xc28`, либо не измените регистр `E40`, либо не выключите устройство.

Второй параметр команды `uncctl -Y` можно снабдить пробелами для лучшей читаемости; при этом, естественно, параметр придётся заключить в кавычки, чтобы командный интерпретатор не разбил его на несколько отдельных параметров при передаче программе `uncctl`. Кроме того, для задания номера регистра можно использовать шестнадцатеричную систему, начав параметр с символа `x`. Например, вышеприведённая команда может быть записана и так:

```
uncctl -Y x28 "f0 00 00ff 0101 0e28"
```

Команда `uncctl -I` подключается к устройству на чтение потока ввода. Если в *регистре пользовательской конфигурации* установлен соответствующий бит, программа будет печатать в поток стандартного вывода («на экран») все сообщения, генерируемые устройством. В этом режиме программа работает до тех пор, пока устройство не окажется отключено, либо пока пользователь не прервёт работу программы средствами операционной системы (например, нажав Ctrl-C). В настоящее время эта опция не работает под Windows.

Команда `uncctl -K` очищает кольцевой буфер сообщений в основной структуре состояния устройства.

Команда `uncctl -J` сбрасывает всю текущую информацию, связанную с шиной 1-Wire, и переинициализирует соответствующую подсистему.

Команда `uncctl -D` позволяет выдать полное содержимое EEPROM и используется в отладочных целях. В частности, с помощью этой команды можно узнать, какие значения хранятся в регистрах `E00`–`E47` (учтите, что байты, составляющие все регистры устройства, хранятся в EEPROM в порядке Little Endian, то есть младший байт идёт первым, старший — последним).

Совместно со всеми перечисленными командами можно использовать опцию `-t` с целочисленным параметром, задающим таймаут на шине USB в миллисекундах. По умолчанию этот таймаут равен `5000`. Не используйте эту опцию, если только вы не уверены, что она вам нужна. При работе под ОС Windows эта опция игнорируется.

Программа `uncctl` имеет также некоторые возможности, не перечисленные здесь в связи с невозможностью их использования с устройствами серии UNC001.

5.3. Разрешение пользовательского доступа с помощью программы `unc_chown` (только для ОС Unix)

По умолчанию в системах семейства Unix (в том числе Linux) осуществить доступ к устройствам USB можно только с правами системного администратора (пользователя `root`), однако при работе с контроллерами UNC0xx это может быть неудобно и не вполне безопасно, в особенности если доступ к контроллерам осуществляет программа с графическим пользовательским интерфейсом. Решить проблему позволяет программа `unc_chown`, которая сканирует шины USB в поисках устройств UNC0xx и в зависимости от настроек меняет идентификаторы владельца/группы и права доступа для соответствующих файлов в псевдодиректории `/dev/bus/usb`. Запуск программы можно осуществлять средствами демона `udevd`, например, при подключении новых USB-устройств с соответствующими значениями VID/PID; можно запускать программу с определённой периодичностью средствами `crond`; наконец, можно установить программу в системе с битом SUID с тем, чтобы пользователи могли сами её запустить (в этом режиме программа отказывается обрабатывать параметры командной строки, так что запускающий её пользователь может получить только результат, предусмотренный системными настройками).

Настройки программы позволяют для каждого пользовательского идентификатора устройств UNC0xx указать предполагаемого владельца, группу и права доступа. Конфигурация задаётся последовательностью из одной или больше *строфы*, каждая строфа состоит ровно из четырёх параметров, разделённых символом двоеточия; сами строфы отделяются друг от друга произвольными пробельными символами, включая собственно пробелы (удобно при использовании командной строки) и переводы строк (удобно при использовании файлов). Первый параметр каждой строфы задаёт пользовательский идентификатор устройства (в виде шестнадцатеричного числа), либо символ «`*`» для обозначения всех остальных (не упомянутых ранее) идентификаторов; такая строфа должна быть последней в конфигурации. Второй и третий параметры задают соответственно пользователя и группу; оба параметра могут быть заданы в виде имени либо в виде числового идентификатора (соответственно `uid` и `gid`). Последний, четвёртый параметр задаёт права доступа к файлу устройства; права задаются в виде восьмеричного числа. Любой параметр, кроме идентификатора, может быть опущен (подразумевается отсутствие необходимости изменения соответствующего атрибута файла). Например, конфигурация

```
1333:::666  2444:john::600  *::unc:660
```

указывает, что устройство с идентификатором `1333` должно быть доступно всем пользователям системы (права доступа `666`), устройство с идентификатором

2444 должно быть доступно только пользователю john, а все прочие устройства должны быть доступны всем пользователям, входящим в группу unc.

Программа может получить свою конфигурацию:

- непосредственно из командной строки;
- из файла, имя которого указано в командной строке;
- из файла /etc/unc_chown.conf;
- либо может быть использована конфигурация по умолчанию — строка *::unc:660.

Параметр командной строки должен быть ровно один. Если первым символом параметра является символ «/», параметр рассматривается как имя файла, в противном случае — непосредственно как строка конфигурации; так, вышеупомянутую конфигурацию можно задействовать командой

```
unc_chown '1333:::666 2444:john::600 *::unc:660'
```

а можно записать её в файл и заставить программу этот файл прочитать:

```
echo '1333:::666 2444:john::600 *::unc:660' > /tmp/unc
unc_chown /tmp/unc
```

При запуске в режиме SUID/SGID (т. е. если идентификатор пользователя отличается от эффективного идентификатора пользователя, либо идентификатор группы отличается от эффективного идентификатора группы) программа отказывается принимать параметры командной строки, что позволяет установить её в системе с битом SUID.

Если командная строка пуста, программа пытается открыть файл /etc/unc_chown.conf и прочитать конфигурацию из него, если же это не удаётся — используется зашитая в текст программы конфигурация по умолчанию (строка *::unc:660). Для успешной работы этого варианта конфигурации необходимо завести в системе группу unc.

Если в вашей системе присутствует сервис udevd, вы можете настроить запуск unc_chown при каждом появлении в системе нового устройства с VID/PID, совпадающими с VID/PID устройств UNC0xx. Для этого добавьте в файл локальных правил (обычно это /etc/udev/rules.d/99-local.rules) следующий текст (в виде **одной строки**):

```
ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="05df", SUBSYSTEMS=="usb",
ACTION=="add", PROGRAM="/usr/local/sbin/unc_chown"
```

При отсутствии udevd (например, в дистрибутиве Openwall GNU/*/Linux) вы можете использовать для периодического запуска программы сервис crond; откройте на редактирование cron-файл администратора командой

```
crontab -eu root
```

и вставьте в него, например, следующую строку:

```
* * * * *   /usr/local/sbin/unc_chown
```

Наконец, вы можете разрешить пользователям системы самим требовать приведения прав доступа к устройствам UNC0xx в соответствие с системными настройками; для этого воспользуйтесь командой

```
chmod 4755 /usr/local/sbin/unc_chown
```

Это позволит запустить программу из сеанса работы любого пользователя системы, причём сама программа при этом будет обладать полномочиями администратора. Поскольку в этом режиме программа отказывается обрабатывать аргументы командной строки, установленные при этом права будут соответствовать системным настройкам, на которые обычный пользователь не может никак повлиять.

VI. Расширенные возможности прошивки

Регистры E00–E47, называемые *расширенными*, в нынешней версии прошивки могут быть использованы:

- при организации автономной работы устройства — для задания усложнённых последовательностей действий, которые устройство выполняет без вмешательства управляющего компьютера;
- при работе с шиной 1-Wire — для хранения кодов известных «ключей» iButton, а также для настройки реакций на изменения температуры, измеряемой термодатчиками.

Каждый E-регистр состоит из восьми байтов. Способ интерпретации значения, занесённого в E-регистр, определяется значением его старшего байта. Поддерживаемые значения приведены в табл. 6.1.

6.1. Последовательности действий

Регистры E00–E47 позволяют задавать более сложные варианты реакции на события и даже организовывать простейшие «программы» — последовательности, состоящие из нескольких различных действий с заданными временны́ми промежутками между ними.

«Сложные действия» делятся на последовательности действий, действия с задержкой и условные действия. Для задания сложного действия подходит любой из расширенных регистров.

Для задания последовательности из пяти действий в одном E-регистре следует этот регистр рассматривать как последовательность из 16 полубайтов (обратите внимание, что каждый полубайт соответствует ровно одной шестнадцатеричной цифре). Самый старший полубайт (полубайт № 15) устанавливается в

Таблица 6.1: Возможные значения старшего байта E-регистра

Старший байт (hex)	Способ использования регистра
00–2F	Идентификатор «таблетки» iButton; конкретное значение старшего байта определяет номер E-регистра, в котором содержится «сложное действие», выполняемое при появлении «таблетки» с этим идентификатором. Структура регистра показана на рис. 6.1 (a).
60	Идентификатор произвольного ведомого устройства 1-Wire. В настоящее время применяется только для устройств ввода/вывода на основе микросхемы DS2413 совместно с кодами действий 9xx. Структура регистра аналогична (рис. 6.1 (a))
70	Идентификатор датчика температуры; один или более следующих регистров должны начинаться с байта 7e и содержать значения верхней и нижней границ температурного диапазона, а также коды действий, выполняемых при пересечении этих границ. Структура регистра аналогична.
7E	Информация о границах температурного диапазона, а также коды действий, выполняемых при пересечении этих границ. Предыдущий регистр должен иметь код 70 или 7E. Структура регистра показана на рис. 6.1 (b).
7F	Регистр не используется, но в следующих за ним могут находиться идентификаторы ведомых устройств 1-Wire.
E0–EF	Регистр содержит последовательность из пяти кодов действий, причём младший полубайт старшего байта используется как часть первого кода действия. Структура регистра показана на рис. 6.1 (c).
F0	Регистр содержит описание «сложного действия». Структура регистра показана на рис. 6.1 (d).
F4–F7	Регистр содержит описание «условного действия». Структура регистра показана на рис. 6.1 (e).
FF	Регистр не используется.

значение E, чтобы показать, что данный регистр содержит последовательность действий. Следующие три полубайта (14, 13 и 12, то есть младший полубайт старшего байта и оба полубайта следующего за ним) задают код первого из пяти действий; полубайты 11, 10 и 9 — код второго действия, и т. д.; код послед-

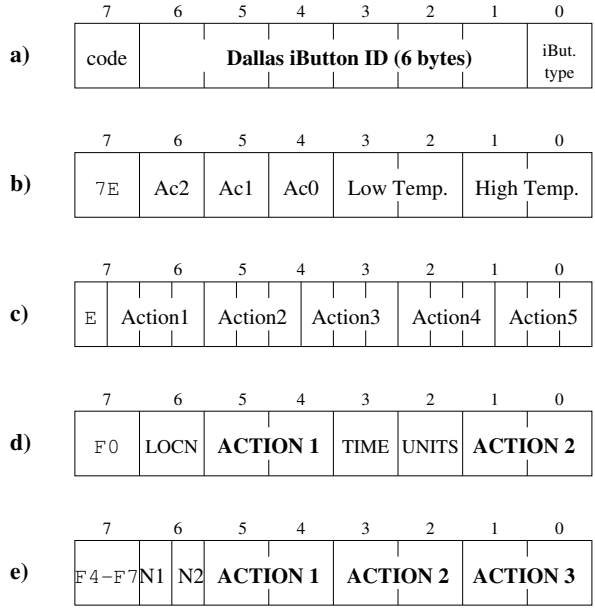


Рис. 6.1: Структура E-регистров при различных способах использования

него, пятого действия в последовательности задаётся тремя младшими (2, 1, 0) полубайтами регистра.

Например, если дать команду

```
uncctl -Y 47 "e 0aa 1aa 2aa cff a00"
```

то регистр `E47` будет соответствовать последовательности действий `x0aa` (выключить все линии основной группы), `x1aa` (выключить все линии первой дополнительной группы), `x2aa` (выключить все линии второй дополнительной группы), `xcff` (отменить все отложенные действия, если таковые есть) и `xa00` (установить значение регистра FSM0 в `0`). Все эти действия будут выполнены последовательно одно за другим, если тем или иным способом приказать устройству выполнить действие `xe2f` (выполнение регистра `E47`).

С другой стороны, команда

```
uncctl -Y 46 "e e28 e29 e2a e2b e2c"
```

занесёт в регистр `E46` последовательность, состоящую из выполнения один за другим действий, занесённых в регистры `E40, E41, E42, E43` и `E44`, которые, в свою очередь, могут быть «сложными».

6.2. Действия с задержкой

Для задания действия с задержкой старший байт (байт № 7) избранного E-регистра должен содержать значение `0xf0`. Байт № 6 при этом используется для задания номера строки в таблице отложенных действий (см. ниже); если сомневаетесь, оставьте его нулевым.

Действие с задержкой состоит из трёх параметров:

- кода действия, которое необходимо выполнить немедленно;
- величины временно́й задержки;
- кода действия, которое необходимо выполнить по истечении задержки.

Возможные типы кодов действий приведены на стр. 17.

Код действия, подлежащего немедленному выполнению, заносится в 5-й и 4-й байты `E`-регистра, причём младший байт кода действия заносится в 4-й, старший — в 5-й байты регистра. Аналогичным образом код действия, подлежащего выполнению после задержки, заносится два самых младших байта: младший байт кода действия заносится в 0-й, старший — в 1-й байты регистра.

Задержка формируется из двух чисел, первое из которых (хранящееся в 3-м байте регистра) задаёт размер задержки, а второе (в 2-м байте) — единицы измерения. Поддерживается измерение времени в *циклах* (код `0`; длительность цикла *приблизительно* равна 0,01 секунды), *секундах* (код `1`), минутах (код `2`) и часах (код `3`). В зависимости от значения 3-го байта регистра количественное значение задержки, указанное во 2-м байте, умножается, соответственно, на 1, 100, 6000 и 360000. Необходимо учитывать, что длительность цикла установлена *приближенно*; использовать устройство в качестве точного таймера не представляется возможным.

Рассмотрим пример. Занесём в регистр `E19` ($19_{10} = 13_{16}$) действие с задержкой, которое предписывает немедленно выполнить код действия `00ff` (переключение выходных линий в противоположное состояние), после чего сделать задержку на 2 секунды и выполнить действие с кодом `0e13`, то есть *снова выполнить всё, что предписано регистром E19*. Соответствующая команда выглядит так:

```
uncctl -Y x13 "f000 00ff 0201 0e13"
```

Если теперь предписать устройству выполнить действие с кодом `e13` (то есть «выполнить действие, описанное в регистре `E19`»), что можно сделать командой

```
uncctl -A xe13
```

то устройство переключит все четыре основные коммутируемые цепи в противоположное положение и установит задержку в 2 с., после которой вновь выполнит действие `e13`, что опять приведёт к переключению всех цепей и установке задержки, и так далее. Таким образом, устройство будет с интервалом в 2 с.

переключать состояние выходных цепей на противоположное, и так до тех пор, пока вы не отмените очередное «действие с задержкой», предписав исполнить код действия `cff` или `c13`, либо не измените содержимое регистра `E19`.

Рассмотрим более сложный пример. В регистры `E10-E13` занесём немедленное действие «включить очередную выходную линию, выключив все остальные», задержку 3 с. и отложенное действие «выполнить следующий регистр в цепочке» (для регистра `E13` следующим будет `E10`). В регистр `E14` занесём немедленное действие «выполнить регистр `E10`», задержку 5 минут и отложенное действие «отменить все отложенные действия». Это можно сделать следующими командами:

```
uncctl -Y x0a "f000 00a9 0301 0e0b"
uncctl -Y x0b "f000 00a6 0301 0e0c"
uncctl -Y x0c "f000 009a 0301 0e0d"
uncctl -Y x0d "f000 006a 0301 0e0a"
uncctl -Y x0e "f000 0e0a 0502 0cff"
```

Если теперь выполнить действие `e0e` (то есть «выполнить действие, описанное регистром `E14`»), устройство начнёт с интервалом в 3 секунды по кругу включать линии **A**, **B**, **C**, **D**, **A** и т. д.; через 5 минут после начала работы сработает отложенное действие `cff`, отменяющее все отложенные действия, и устройство прекратит переключать линии (при этом одна из линий останется включена).

Не меняя значений уже установленных регистров, выполним теперь команды

```
uncctl -Y x0f "f000 0cff 0000 0f00"
uncctl -R 0 600e0f0a
```

Теперь при замыкании входной цепи № 0 устройство будет выполнять действие с кодом `e0a`, а при её размыкании — действие `e0f`. Первое из них («выполни регистр `E10`») запустит уже знакомую нам последовательность включений линий по очереди; второе из них выполнит регистр `E15`, который мы только что установили; в нём отсутствует задержка и код отложенного действия равен `0f00` (отсутствие действия), что касается действия, выполняемого немедленно, то его код — `cff`, то есть «отмена всех отложенных действий». Таким образом, замыкание цепи 0 будет включать процесс переключения цепей, а размыкание — выключать его.

Устройство способно в каждый момент времени «помнить» десять «отложенных действий», каждое со своей задержкой и со своим кодом действия. Попытка установить больше отложенных действий ни к чему не приведёт: соответствующее действие будет просто проигнорировано. Для хранения кодов отложенных действий и времени, оставшегося до их исполнения, устройство использует *таблицу отложенных действий* из 10 строк. В случае, если байт № 6 регистра вы оставили нулевым, отложенное действие размещается в первой свободной строке таблицы. Вы также можете установить этот байт в значения `0x10`, `0x11`, ..., `0x19`, что будет означать использование соответственно строки таблицы с номером `0`,

1, ..., 9; в случае, если соответствующая строка уже занята, старое отложенное действие заменяется новым. Например, команда

```
uncctl -Y x20 "f019 0001 0502 0002"
```

занесёт в регистр E32 (20_{16}) сложное действие с задержкой, состоящее из включения первой линии основной группы (0001), задержки на пять минут (0502) и выключения первой линии основной группы (0002), причём отложенное действие размещается в строке № 9 таблицы отложенных действий. Если теперь выполнить этот регистр:

```
uncctl -A xe20
```

то линия будет включена и устройство начнёт обратный отсчёт пяти минут до её выключения; повторное выполнение той же команды снова включит линию (на случай, если кто-то её выключил) и начнёт отсчёт с начала, заместив старое отложенное действие (в строке № 9) на новое.

Не рекомендуется смешивать в настройках одного устройства отложенные действия без задания размещения и отложенные действия с заданным размещением: при этом достаточно трудно отследить ситуацию, когда используемая (фиксированная) позиция в таблице оказывается занята действием, заданным без фиксированного размещения, и две разные «программы» могут таким образом повлиять друг на друга.

6.3. Условные действия и конечные автоматы

На практике часто возникают задачи, в которых реакция устройства на те или иные события должна зависеть от некоего *состояния*, чем бы таковое ни являлось. Наиболее очевидные примеры таких задач связаны с охранными системами: например, реакция на срабатывание датчиков движения, очевидно, должна быть совершенно различной в зависимости от того, поставлено ли помещение «на охрану» или снято с неё, причём режим охраны может иметь несколько уровней, и так далее. Аналогичным образом автоматизированное управление освещением в квартире может работать по-разному в режиме присутствия хозяев, в режиме их кратковременного отсутствия и в режиме длительного отсутствия. Можно привести и другие примеры.

Благодаря поддержке FSM-регистра[1] (см. § 4.3) устройства UNC0xx позволяют построить логику автономной работы, учитывающую текущее *состояние*, которое может быть в любой момент изменено. Вы можете задействовать до четырёх независимых *конечных автоматов*, каждый из которых может находиться в одном из 16 состояний.

Когда при выполнении кода действия x0ERR, где RR — номер E-регистра, устройство обнаруживает, что в старшем байте указанного E-регистра находится число от F4 до F7, содержимое соответствующего регистра воспринимается

[1] Аббревиатура FSM означает *finite state machine*, т. е. конечный автомат.

как задающее условное действие, причём F4 предполагает выполнение одного из трёх заданных действий в зависимости от значения регистра FSM0, F5 — в зависимости от FSM1, F6 — в зависимости от FSM2 и, наконец, F7 — в зависимости от FSM3.

Байт № 6 E-регистра в этом случае задаёт два «граничных» значения, каждое от 0 до 16, которые мы условно назовём N_1 и N_2. Далее в регистре размещаются двухбайтные коды трёх действий. Если значение выбранного FSM-регистра не превышает N_1, выполняется первое из них (заданное байтами 5 и 4); если значение FSM-регистра больше, чем N_1, но не больше, чем N_2, будет выполнено второе (байты 3 и 2); наконец, если значение превышает N_2, выполняется третье из заданных действий (байты 1 и 0).

Пусть, например, в регистр E12 записано значение F5 3A 0E27 0CFF 0055. Если теперь выполнить код действия E0C, тем самым заставив устройство исполнить регистр E12, то дальнейшее будет зависеть от значения в регистре FSM1: если это значение не превышает 3, устройство выполнит действие 0E27 (то есть исполнит регистр E39), если значение окажется от 4 до 10 включительно, будет выполнено действие 0CFF (отмена всех отложенных действий, если таковые есть), если же значение окажется превышающим 10, то будет выполнено действие 0055, означающее включение всех линий (реле) основной группы.

Изменять значения FSM-регистров позволяют коды действий, имеющие старший байт 0A, при этом младший байт кода действия используется для задания номера FSM-регистра (биты 5 и 4), а также нового значения FSM-регистра (биты 3, 2, 1 и 0). Например, выполнение действия 0A29 установит регистр FSM2 в значение 9, тогда как команда 0A0B установит регистр FSM0 в значение B.

Рассмотрим пример. Пусть к коммутируемой цепи A подключён электромагнит дверного замка. Устройство оснащено модулем UNC001/in, который обеспечивает работу входных (сигнальных) линий in0–in3. Внутри помещения расположены кнопка открывания двери, подсоединённая к линии in0, и тумблер включения-выключения режима «дверь заперта», который подсоединён к линии in1. С наружной стороны двери расположены две кнопки B1 и B2, подсоединённые, соответственно, к линиям in2 и in3.

Логика работы в нашем примере будет организована следующим образом. Пока тумблер находится в разомкнутом положении, считается, что дверь отперта; при нажатии кнопки открывания внутри помещения с электромагнитного замка напряжение снимается на три секунды. Для открытия двери извне помещения необходимо выполнить более сложное действие: нажать кнопку B1, выждать, не отпуская её, не менее двух секунд, нажать и отпустить кнопку B2, и только после этого отпустить кнопку B1.

После перевода тумблера в замкнутое положение дверь считается запертой. Для открывания её изнутри теперь необходимо удерживать внутреннюю кнопку не менее трёх, но не более семи секунд. Внешние кнопки теперь не работают вообще.

Чтобы реализовать описанную логику, условимся прежде всего, что значение регистра FSM0, равное нулю, будет означать, что дверь отперта, а для «запирания» двери мы применим значение 7. Заставим тумблер на линии in1 менять это состояние:

```
uncctl -R 1 000A0007
```

Как видим, при размыкании линии in1 будет выполняться действие `A00`, обнуляющее FSM0, а при замыкании будет выполнено `A07`, заносящее в FSM0 значение 7.

E-регистры будем использовать по порядку «сверху вниз», начиная с E47 (причины этого указаны в § 6.7). Открывание двери подразумевает, что линию A нужно обесточить, выждать три секунды и снова включить линию A. Реализуем это в регистре E47:

```
uncctl -E 47 'F000 0002 0301 0001'
```

Если теперь выполнить действие `E2F` (например, дать команду `uncctl -A xE2F`), именно такая последовательность будет исполнена ($\mathtt{2F}_{16} = 47_{10}$).

Подготовим теперь к использованию кнопку внутри помещения. Реакция на её нажатие и отпускание зависит от состояния, так что придётся задействовать очередные E-регистры. При нажатии этой кнопки, если мы находимся в режиме «дверь отперта», нам достаточно немедленно исполнить регистр E47; если же дверь «заперта», нам следует выждать три секунды, неким образом разрешить открывание двери по отпусканию кнопки, выждать ещё четыре секунды, и если кнопка так и не была отпущена, снова запретить открывание двери. Для реакции на отпускание кнопки задействуем регистр FSM1: при нулевом значении этого регистра размыкание in0 не будет иметь никакого эффекта, тогда как при значении 7 такое размыкание произведёт открытие двери. Наконец, есть ещё один вариант: если кнопка была нажата, трёх секунд ещё не прошло, а кнопку уже отпустили, необходимо забыть про то, что мы собирались разрешать открывание двери. Это состояние закодируем значением F.

Забегая вперёд, скажем, что разрешение открывания двери будет реализовано в виде отложенного запуска регистра E44, именно такой запуск нужно отменить, если мы находимся в состоянии F. Используем для этого регистр E46:

```
uncctl -E 46 "f567 0f00 0e2f 0c2c"
```

Как видим, выполнение действия `e2e`, «запускающее» этот регистр, при значениях FSM1, меньших либо равных 6, не делает ничего (`0f00`, при значении 7 запускает регистр E47 (`0e2f`), при остальных значениях отменяет отложенное выполнение E44. На событие размыкания линии in0 теперь можно «повесить» именно это действие.

Для обработки замыкания in0 нам потребуется запрограммировать последовательность «подождать — разрешить — подождать — запретить». Для этого мы используем два регистра — E45 и E44:

```
uncctl -E 45 'f000 0a1f 0301 0e2c'
uncctl -E 44 'f000 0a17 0401 0a10'
```

Как видим, E45 заносит F в FSM1 и планирует запуск E44 через три секунды, а E44 заносит в FSM1 значение 7 и планирует через четыре секунды действие по занесению туда нуля. Вспомним теперь, что всё это следует делать лишь в случае, если мы находимся в состоянии «дверь заперта», в ином же случае мы просто отпираем дверь. Введём условное действие в регистре E43:

```
uncctl -E 43 'f467 0e2f 0e2d 0f00'
```

(если в FSM0 у нас число от 0 до 6, открываем дверь (0E2F), если ровно 7 — запускаем вышеописанную последовательность (0E2D), если больше — не делаем ничего, всё равно так быть не может). Напомним, что 43 в шестнадцатеричной системе будет 2B.

Имея готовые последовательности для замыкания и размыкания in0, мы можем теперь задать регистр реакции для in0 (выполнение 0e2b при замыкании и 0e2e при размыкании):

```
uncctl -R 0 000e2e2b
```

Вернёмся теперь к внешним кнопкам B1 и B2. Всё начинается с нажатия кнопки B1, то есть замыкания цепи in2; в случае, если дверь «заперта», нам достаточно просто проигнорировать это нажатие. Для реализации логики, описанной для этих кнопок, мы задействуем регистр FSM2. Нажатие B1 переведёт FSM2 в состояние 1 и запланирует через две секунды перевод в состояние 2, нажатие B2 переведёт FSM2 в состояние 3, если оно было 2, в противном случае — вернёт FSM2 в исходное состояние 0. Отпускание B2 переведёт FSM2 из состояния 3 в состояние 4 (иначе в исходное), и, наконец, отпускание B3 в случае, если состояние FSM2 было равно 4, откроет дверь, при этом в любом случае вернёт FSM2 в исходное состояние:

```
uncctl -E x2A 'f467 0e29 0f00 0f00'
uncctl -E x29 'f000 0a31 0201 0e28'
uncctl -E x28 'f601 0a30 0a32 0a30'
uncctl -E x27 'f612 0a30 0a33 0a30'
uncctl -E x26 'f623 0a30 0a34 0a30'
uncctl -E x25 'f000 0e24 0100 0e23'
uncctl -E x24 'f634 0f00 0e2f 0f00'
uncctl -E x23 'f000 0a30 0000 0f00'
uncctl -R 2 000e252a
uncctl -R 3 000e2627
```

Подробный анализ приведённых кодов оставляем читателю для самостоятельного упражнения.

Безусловно, описанные возможности достаточно примитивны; для решения более сложных задач следует использовать управляющий компьютер.

6.4. Работа с шиной 1-Wire

Действия устройства с шиной 1-Wire настраиваются третьим байтом *регистра пользовательской конфигурации* (см. табл. 4.2 на стр. 13). Если этот байт равен нулю, никаких действий с шиной 1-Wire не производится. Для осуществления какой-либо работы с 1-Wire необходимо установить как минимум старший бит этого байта (23-й бит регистра) в единицу; при этом регулярно (приблизительно шесть раз в секунду) будет выполняться сканирование шины 1-Wire на предмет подключённых устройств.

Прошивка способна работать не более чем с шестью ведомыми устройствами, подключёнными к 1-Wire; при обнаружении бóльшего их количества «лишние» устройства будут просто проигнорированы, причём невозможно предсказать, какие именно будут проигнорированы, а какие учтены. Для шести ведомых устройств прошивка запоминает их идентификаторы (8-байтные массивы), что позволяет фиксировать факт подключения и отключения таких устройств. Для запоминания используется область памяти *расширенного состояния*.

22-й и 21-й биты регистра пользовательской конфигурации позволяют включить выдачу текстовых сообщений, соответственно, о подключении и отключении 1-Wire-устройств; эти сообщения состоят из буквы «`W`», знака «`+`» или «`-`» (соотв., для подключения и отключения) и 16 шестнадцатеричных цифр, составляющих идентификатор устройства.

20-й бит регистра пользовательской конфигурации предписывает прошивке при обнаружении датчиков температуры `DS18x20` регулярно считывать с них показания температуры; прочитанные показания заносятся в область *расширенного состояния*.

19-й бит регистра пользовательской конфигурации предписывает сканирование регистров `E00`, `E01`, `E02` и т. д. на предмет идентификаторов «таблеток» iButton при обнаружении подключения такой «таблетки» (устройства с кодом типа `0x01`). Подробно эта функциональность описана в § 6.5.

18-й бит регистра пользовательской конфигурации включает обработку верхних и нижних границ температурных диапазонов для датчиков температуры. Подробности приведены ниже в § 6.6.

Например, команда

```
uncctl -G 00fc0001
```

задействует все вышеперечисленные возможности прошивки по обработке 1-Wire (байт `0xfc` состоит из шести взведённых единиц и двух нулей).

Биты регистра пользовательской конфигурации с 15 по 12 (старший полубайт второго байта регистра) задаёт величину *задержки перед констатацией отключения (потери) ведомого устройства*. Значение, указанное в регистре, умножается на 16, к результату добавляется 15; полученный результат используется как число циклов устройства UNC0xx, по прошествии которых с момента потери связи с ведомым следует констатировать его отключение и убрать его

идентификатор из таблицы. Выполнение указанной задержки преследует две цели. Во-первых, временная потеря связи с ведомым устройством может означать как его отключение, так и временный сбой на линии связи, тем более вероятный, чем больше длина линии. Во-вторых, для ряда задач целесообразно дать управляющему компьютеру возможность прочитать информацию об идентификаторе и статусе ведомого устройства в течение некоторого времени после его отключения.

Следует отметить, что величина задержки в любом случае не будет меньше определённого минимального значения (в текущей версии прошивки — около 50 циклов, т. е. 0,5 секунды); если отключения фиксировать раньше, работа окажется затруднена даже на очень коротких линиях 1-Wire. Кроме того, **специальное значение `f` означает *никогда* не считать ведомые устройства потерянными**. Такое может понадобиться при использовании фиксированного набора ведомых устройств, относительно которых интересен их последний прочитанный статус (вне зависимости от того, насколько давно он был прочитан). При этом в зависимости от потребностей приложения можно периодически принудительно сбрасывать список подключённых ведомых устройств (например, командой `uncctl -J`).

Например, команда

```
uncctl -G 00fce001
```

задействует все имеющиеся возможности 1-Wire и дополнительно устанавливает максимально возможную величину задержки отключения (239 циклов, то есть почти 2,5 секунды), а команда

```
uncctl -G 00fcf001
```

вообще запрещает считать ведомые устройства отключёнными.

6.5. Реакция на появление «таблеток» iButton с заранее заданными кодами

Устройство обладает способностью помнить коды «таблеток» iButton и при появлении на шине 1-Wire новой «таблетки» сличать её код с кодами, записанными в E-регистрах. В случае совпадения выполняется «сложное действие», которое необходимо заранее записать в один из E-регистров. Коды «таблеток» должны храниться в непрерывной области регистров, начинающейся с регистра `E00`; так, если вы хотите, чтобы ваше устройство опознавало коды двенадцати различных «таблеток», вам придётся задействовать регистры `E00`, `E01` ... `E11`. Максимальное количество «таблеток» составляет, таким образом, 47 (один регистр необходимо оставить для описания «сложного действия»), но при этом не останется места для других «сложных действий», а также и для реакций на изменение температуры.

В старший байт E-регистра, хранящего код «таблетки», заносится номер E-регистра, содержащего описание «сложного действия», то есть шестнадцатеричное число от 00 до 2F. Следующие шесть байт используются для хранения идентификатора «таблетки»; в младший байт заносится код типа устройства 1-Wire, который для «таблеток» обычно равен 01. Отметим, что часто говорят о **восьми**байтном коде устройств 1-Wire; речь в этом случае идёт о 64-битном числе, в котором младший байт — это код типа устройства (для «таблетки» 01), а старший байт представляет собой контрольную сумму, то есть однозначно определяется остальными байтами числа. Например, можно говорить о «таблетке» с номером B3000008971E8B01; здесь 01 — код типа, 000008971E8B — уникальный заводской идентификатор устройства, и B3 — контрольная сумма. В E-регистр не заносится байт контрольной суммы, на его месте находится байт, определяющий способ использования E-регистра (в данном случае — байт со значением от 00 до 2f).

Например, следующие команды:

```
uncctl -Y 0 2e000008971e8b01
uncctl -Y x2e "f000 0001 0301 0002"
```

заносят в регистр E00 идентификатор iButton с серийным номером 000008971E8B, снабженный указанием выполнять при появлении такого идентификатора действие, заданное регистром E46, а в регистр E46 ($2E_{16}$) — указание включить первую из коммутируемых линий (0001), подождать три секунды (0301) и выключить её (0002). Если к первой коммутируемой линии подсоединить электромагнит дверного замка, устройство, настроенное таким образом, сможет выполнить роль его контроллера. Если небходимо опознавать больше одного ключа iButton, используйте регистры E01, E02 и т. д. (идущие подряд).

Идентификатор «таблетки» обычно выгравирован на её контактной поверхности. Если идентификатор плохо читается, вы можете узнать его с помощью устройства UNC01x: подключите «таблетку» к шине и, не отключая её, дайте команду

```
uncctl -Q
```

В конце выдачи команды вы увидите строку вида

```
1w: b3000008971e8b01
```

Это и есть идентификатор «таблетки», включая контрольную сумму. Отбросив первые две цифры (в данном случае b3), вы получите нужные вам семь байт. Обратите внимание, что именно этот номер «таблетки» мы использовали в вышеприведённом примере (заносили его в E00).

Устройство UNC0xx выполняет сканирование E-регистров в поисках идентификатора «таблетки» или другого ведомого устройства 1-Wire, начиная с регистра E00, в порядке возрастания номеров регистров; поиск заканчивается, когда будет обнаружен регистр, в старшем байте которого записано число, бóльшее 7F (например, незадействованный регистр, старший байт которого равен FF, или регистр, хранящий «сложное действие», старший байт в этом случае равен F0).

6.6. Реакция на события, связанные с температурой

Если на вашей шине 1-Wire присутствуют датчики температуры DS18B20 или DS18S20, то вы можете задать для каждого такого датчика диапазон температур, выход за границы которого (пересечение нижней границы в направлении вниз, а верхней — в направлении вверх) вызовет выполнение того или иного действия (в том числе сложного).

Для настройки таких действий можно использовать два или несколько E-регистров, идущих подряд, например, E02:E03, E13:E14:E15 и т. д. В младший из выбранных регистров заносится 1-Wire-идентификатор датчика так же, как это делалось для «таблеток» в предыдущем параграфе, только старший байт устанавливается в специальное значение 70; младший байт регистра, таким образом, должен стать равен 0x10 или 0x28 в зависимости от типа датчика.

Последующие регистры заполняются следующим образом. В старший байт заносится специальное значение 7E, обозначающее «подчинённый» регистр регистровой группы. В следующий (6-й) байт регистра заносится *старший* байт кода действия, используемый при переходе как через верхнюю, так и через нижнюю границы. В 5-й и в 4-й байты заносятся *младшие* байты кода действия, соответственно, для случаев перехода через нижнюю и через верхнюю границы температуры.

В 4-й и 3-й байты регистра заносится двубайтное знаковое целое, соответствующее нижней границе диапазона температуры, а в следующие два байта — аналогичное число, соответствующее верхней границе диапазона температуры. В обоих случаях единицей измерения является $\frac{1}{16}{}^{o}C$ — например, 0xfff8 соответствует $-0{,}5{}^{o}C$, а 0x00f4 — $+15{,}25{}^{o}C$.

Например, следующие команды

```
uncctl -Y 3 "70 000003E622A2 28"
uncctl -Y 4 "7E 0E1A1B 0190 01B0"
```

используют регистровую пару E03:E04, чтобы установить для термодатчика, имеющего идентификатор 000003E622A2 и тип устройства 28, нижнюю границу температуры $25{}^{o}C$, верхнюю — $28{}^{o}C$ (соответственно 0x0190 и 0x01b0), при выходе за которые выполняются действия, заданные регистрами E26 и E27 (соответствующие коды действий 0E1A и 0E1B составлены из байта 0E, используемого в качестве старшего байта кода действия для обоих случаев, и отдельных байтов 1A и 1B).

Более двух регистров используется в случае, если для одного датчика необходимо задать больше одной пары границ. Первый регистр такой последовательности имеет старший байт 70, последующие — 7e.

Учтите, что на практике значение температуры может «проскочить» заданную границу несколько раз подряд. Это не создаёт проблем, если действия, предписанные к выполнению при прохождении границы, будучи выполненными несколько раз, приводят к тем же результатам, что и при однократном выполнении (например, если действие состоит во включении одной из управляемых

линий, то нам совершенно всё равно, сколько раз устройство отработает команду на её включение — один раз или больше).

Кроме того, надёжность шины 1-Wire сравнительно невысока, в связи с чем при очередном опросе датчика ваше устройство может не считать показания температуры или вообще прийти к выводу, что датчик от шины отключён. В обоих случаях показания температуры для данного идентификатора датчика теряются, так что при последующем успешном чтении показаний устройство не может определить, каковы были *предыдущие* показания. Аналогичная ситуация возникает при включении устройства, а также при подключении нового датчика. В версиях прошивки, начиная с 2131003 (и всех более поздних), если текущие показания ниже установленной нижней границы или выше верхней, происходит выполнение заданных действий, как если бы граница была только что пройдена. Отметим, что в более ранних версиях прошивки этого не происходило, в результате чего граница температур могла быть пройдена в тот момент, когда устройство не смогло считать показания или вообще по каким-либо причинам было выключено, и предписанные действия в этом случае не выполнялись.

В случае, если выполнять действие, предписанное температурной границей, несколько раз подряд не годится (например, если это действие — запуск стартёра или ещё что-то подобное), необходимо поставить соответствующие действия в зависимость от одного из FSM-регистров и при выполнении действия изменять значение FSM-регистра. Подробности об использовании FSM-регистров см. в § 6.3.

Узнать идентификатор датчика температуры можно точно так же, как и идентификатор «таблетки», с помощью команды `uncctl -Q` (см. предыдущий параграф).

Необходимо отметить, что сканирование E-регистров в поисках «температурной» регистровой пары производится в таком же порядке, как и при поиске идентификатора «таблетки», то есть начиная с `E00` и до первого регистра, старший байт которого больше `7F`. Таким образом, вы можете произвольно смешивать коды «таблеток» и регистровые последовательности, соответствующие температурным датчикам, но перемежать их неиспользуемыми регистрами и/или регистрами «сложных действий» не следует. Если необходимо вывести из использования один из E-регистров, занесите в его старший байт значение `7F` вместо обычного `FF` — это значение также означает «неиспользуемый регистр», но сканирование регистров на нём не останавливается.

Кроме того, следует помнить, что устройство UNC0xx может работать одновременно не более чем с шестью 1-Wire-устройствами. Безусловно, вы можете задать больше шести регистровых пар или последовательностей, но подключать больше шести датчиков температуры не рекомендуется: неизвестно (и невозможно предсказать), какие из них в каждый момент времени окажутся «видны» управляющему устройству, причём этот набор «видимых» датчиков может непредсказуемо меняться и, как следствие, переходы через заданные границы диапазонов температуры устройство может «прозевать».

Более того, если предполагается также работа с «таблетками», количество одновременно подключённых температурных датчиков следует ограничить пятью, чтобы дать возможность устройству работать также и с «таблеткой», когда таковая окажется подключена.

6.7. Рекомендации по распределению регистров Exx

Обычно сложно предвидеть заранее, сколько регистров потребуется для хранения кодов «таблеток», а сколько — для хранения «сложных реакций».

Как уже говорилось, поиск регистров, в которых хранятся коды «таблеток», а также регистровых пары для термодатчиков начинается с `E00`. Предсказать, где эта область регистров будет заканчиваться, сложно — вам в любой момент может потребоваться ещё одна «таблетка» или ещё один термодатчик. С другой стороны, сложно предсказать также и то, сколько вам потребуется «расширенных реакций». Чтобы избежать лишних перестраиваний регистров, мы рекомендуем проводить заполнение `E`-регистров описаниями «расширенных реакций», начиная с регистра `E47` в сторону уменьшения номеров (то есть заполнять регистры `E47`, `E46`, `E45` и так далее, по мере надобности).

Если возникла необходимость освободить регистр, это можно сделать, занеся в него восемь байтов `0xff`, например:

```
uncctl -Y 14 "ffff ffff ffff ffff"
```

Однако такой способ не всегда пригоден для избавления от кода таблетки или температурной регистровой пары, поскольку при этом будет потеряна возможность опознания не только того устройства таблетки, код которого был затёрт, но и тех, коды которых содержатся в последующих регистрах (напоминаем, устойство сканирует регистры в поисках кода появившейся «таблетки» или термодатчика, начиная с `E00`, и заканчивая тогда, когда будет обнаружен регистр, **не** содержащий код таблетки и не являющийся частью регистровой пары). В этой ситуации правильней будет применить код `7F`, а не `FF`; он также обозначает неиспользуемый регистр, но поиск на этом коде не останавливается:

```
uncctl -Y 14 "7fff ffff ffff ffff"
```

VII. Интерфейс прикладного программирования (языки Си и Си++)

7.1. Общее описание

В состав программного обеспечения для ПК, обеспечивающего работу с устройствами UNC001 и UNC01x, входит набор модулей, реализующих функции для связи с устройством через интерфейсы USB и RS485. Модули объединяются в библиотеку `unc0xx` и снабжены заголовочным файлом `unc0xx.h`. Исходный текст модулей находится в файлах `unc0xx.c`, `unc0xx_usb.c` и

unc0xx_rs485.c. При сборке программного обеспечения создаётся статическая библиотека libunc0xx.a; рекомендуется при сборке ваших программ использовать именно этот библиотечный файл, хотя возможно, разумеется, и прямое использование модулей, составляющих библиотеку. В версии для Windows в библиотеку также включается модуль hidapi; в версии для Linux функции модуля unc0xx_usb используют возможности библиотеки libusb, входящей практически во все дистрибутивы Linux.

Для использования функций библиотеки unc0xx в программе, написанной на языке Си или Си++, необходимо в начало исходного текста вставить директиву

```
#include <unc0xx.h>
```

Компиляция отдельных модулей производится с указанием пути к директории, содержащей этот файл (флаг[1] -I), например:

```
gcc -Wall -g -I/home/vasya/unchost -c mymodule.c
```

Финальная сборка проводится с подключением библиотеки unc0xx; при работе в ОС Unix подключается также библиотека libusb, например:

```
gcc -Wall -g -I/home/vasya/unchost myprogram.c   \
    mod1.o mod2.o -L/home/vasya/unchost -lunc0xx \
    `libusb-config --libs`  -o myprogram
```

Помните, что библиотека libusb должна быть подключена **после** библиотеки unc0xx. При работе под Windows библиотека libusb не нужна, поскольку программа использует штатные возможности HID-драйвера Windows, однако при компиляции с использованием MinGW необходимо подключение библиотеки setupapi (флаг -lsetupapi).

Имена всех функций и типов, вводимых библиотекой, начинаются с префикса «unc0xx_»[2] Публичные объекты (функции, глобальные переменные и описания типов), поддерживаемые библиотекой, перечислены ниже.

7.2. Принципы взаимодействия с библиотекой unc0xx

Библиотека содержит ряд функций, специфических для используемого интерфейса (USB или RS485) и используемых для установления соединения с устройством. Результатом установления соединения становится указатель типа struct unc0xx_devhdl *, который в дальнейшем используется для взаимодействия с устройством; функции, используемые для этого взаимодействия, от используемого интерфейса уже не зависят.

[1] Здесь и далее рекомендации даются в предположении, что работа проводится с использованием компилятора gcc; при работе под Windows рекомендуется использование пакета MinGW, что позволит воспользоваться приведёнными рекомендациями с незначительными изменениями.

[2] Более ранняя версия библиотеки, не имевшая средств работы через RS485, называлась uncusb и использовала исторически сложившийся префикс unc001_. Большинство функций новой библиотеки имеют абсолютно такой же интерфейс.

В случае, если предполагается использование USB, перед началом работы необходимо инициализировать библиотеку, обеспечивающую работу с USB; это можно сделать вызовом

```
unc0xx_usb_init();
```

Обратите внимание, что данный вызов инициализирует именно внешнюю библиотеку, обеспечивающую взаимодействие с USB (в случае ОС Unix это `libusb`, в случае Windows — `HidAPI`). В случае, если ваша программа работает с другими USB-устройствами, используя ту же самую внешнюю библиотеку, дважды инициализацию можно не выполнять; сама по себе библиотека `unc0xx` инициализации не требует.

После инициализации следует выполнить сканирование USB-шины в поисках устройств, которыми можно управлять. Это делается с помощью функции `unc0xx_scan()`, которая получает на вход один параметр — «пользовательский идентификатор» нужного устройства (см. стр. 23), либо число 0 для поиска всех устройств UNC0xx, подключённых к компьютеру по USB в настоящий момент. Поскольку обнаружено может быть больше одного устройства, функция формирует односвязный список, состоящий из структур типа `struct unc0xx_item`, и возвращает указатель на первый элемент списка. Если ни одного устройства не обнаружено, возвращается нулевой указатель. Список должен быть впоследствии уничтожен с помощью функции `unc0xx_destroy_list()`. Например:

```
struct unc0xx_item *list;
unc0xx_usb_init();
list = unc0xx_scan(0);
if(list) {

    /* ... работа с устройствами ... */

    unc0xx_destroy_list(list);
} else {
    fprintf(stderr, "No devices found");
}
```

Структура `unc0xx_scan_item` содержит по меньшей мере следующие поля:

```
int                         sernum[3];
int                         version;
struct unc0xx_devhdl *      dev_hdl;
struct unc0xx_scan_item *   next;
```

Элементы массива `sernum` содержат три части серийного номера обнаруженного устройства; в частности, `sernum[2]` равен «пользовательскому идентификатору» устройства. Поле `version` содержит номер версии прошивки устройства, либо число 0, если версия настолько старая, что определить её номер не

представляется возможным (это происходит только с устройствами, проданными ранее июня 2011 года). Поле `dev_hdl` содержит указатель, идентифицирующий установленное с устройством соединение; большинство функций библиотеки требуют указания этого идентификатора для взаимодействия с устройством. Если устройство имеет устаревшую версию прошивки, не поддерживаемую текущей версией библиотеки, это поле будет содержать `NULL`. Наконец, поле `next` представляет собой указатель на следующий элемент в списке обнаруженных устройств, `NULL` для последнего элемента списка.

Например, следующий фрагмент кода напечатает идентификаторы всех обнаруженных устройств, пометив знаком «+» те из них, с которыми библиотека способна работать, и знаком «-» те, версия прошивки которых с библиотекой несовместима:

```
struct unc0xx_scan_item *list, *t;
int cn = 0;
unc0xx_usb_init();
list = unc0xx_scan(0);
printf("Found: ");
for(t = list; t; t = t->next) {
    printf("%x%c ", t->sernum[2], t->dev_hdl ? '+' : '-');
    cn++;
}
printf("\nTotally %d devices found\n", cn);
unc0xx_destroy_list(list);
```

Функция `unc0xx_destroy_list()`, кроме освобождения памяти, выполняет также закрытие соединений с устройствами, перечисленными в списке. Если вам необходимо удалить список, сохранив открытым соединение с одним или несколькими устройствами, занесите `NULL` в поле `dev_hdl` соответствующих элементов списка. Например:

```
struct unc0xx_scan_item *list;
void *dh = NULL;
unc0xx_usb_init();
list = unc0xx_scan(0);
if(list) {
    dh = list->dev_hdl;
    list->dev_hdl = NULL;
    unc0xx_destroy_list(list);
}
```

После выполнения этого фрагмента в указателе `dh` будет находиться идентификатор соединения с устройством, которое в списке обнаруженных устройств оказалось первым, при этом сам список будет удалён и соединения со всеми остальными устройствами, если таковые были, окажутся закрыты.

При работе с устройствами, подключёнными через интерфейс RS485, инициализировать библиотеку не требуется. Сканирование на предмет обнаружения подключённых устройств в этом случае невозможно; пользовательский идентификатор устройства должен быть известен заранее, в противном случае установить соединение с этим устройством не удастся.

Интерфейс RS485 представляется в операционной системе как обычный последовательный порт и имеет соответствующее имя (например, `/dev/ttyS0`, `/dev/ttyUSB0` в Linux и других Unix-системах, `COM1`, `COM4` и т. п. под Windows). RS485 представляет собой шину, поэтому через один порт можно подключить несколько устройств и работать с ними, держа все подключения открытыми. Библиотека `unc0xx` старается избежать повторного открытия одного и того же файла последовательного порта на уровне операционной системы; делается это на основе простого сравнения имён портов (как строк), так что метод нельзя считать совершенно надёжным. В самом деле, один и тот же порт может быть обозначен несколькими формально различными строками (например, `/dev/ttyS0` и `//dev///ttyS0`); однако для практических применений этого обычно достаточно. Несколько открытых соединений с устройствами UNC0xx, использующие один и тот же порт RS485, будут на нижнем уровне разделять между собой дескриптор открытого файла (потока ввода-вывода). От пользователя (на уровне функций библиотеки) этот факт скрыт, но его следует учитывать при разработке программ.

Для установления соединения с устройством UNX0xx через интерфейс RS485 следует использовать функции `unc0xx_rs485_open` или `unc0xx_rs485_probe`. Основное различие между ними в том, что первая только открывает последовательный порт (если он ещё не открыт) и сообщает о готовности, никак не проверяя наличие или отсутствие устройства UNC0xx с заданным идентификатором, тогда как вторая проверяет наличие устройства, запросив его конфигурацию, и, пользуясь полученной информацией, сообщает вызывающему номер версии прошивки устройства. Например, следующий фрагмент

```
struct unc0xx_devhdl* dev_hdl;
dev_hdl = unc0xx_rs485_open("/dev/ttyUSB0", 0, 0x7775);
```

просто попытается открыть порт `/dev/ttyUSB0` и в случае успеха вернёт дескриптор соединения с устройством, даже если никакого устройства на самом деле к порту не подключено; ошибка будет обнаружена лишь при попытке с этим устройством работать. В то же время фрагмент

```
struct unc0xx_devhdl* dev_hdl;
int version;
dev_hdl = unc0xx_rs485_open("/dev/ttyUSB0", 0, 0x7775, &version);
```

отработает успешно только при условии, что к порту подключено устройство с идентификатором 7775, и в этом случае версия прошивки будет записана в переменную `version`.

Закрыть соединение с отдельным устройством позволяет функция `unc0xx_close()`:

```
unc0xx_close(dh);
```

Используя идентификатор открытого соединения, вы можете осуществлять взаимодействие с устройством. В частности, вы можете запросить конфигурацию и состояние устройства с помощью функций `unc0xx_get_fullstate()`, `unc0xx_get_extrastate()` и `unc0xx_get_configuration()`. Заставить устройство выполнить то или иное действие можно при помощи функции `unc0xx_perform_action()`, и так далее. Подробное описание функций библиотеки `unc0xx` приведено в следующем параграфе.

7.3. Справочник по функциям

Функция `unc0xx_usb_init()`

```
void unc0xx_usb_init();
```

Инициализирует библиотеку; необходимо вызвать эту функцию один раз перед началом работы. На самом деле эта функция производит только инициализацию библиотеки `libusb`, так что, если вы используете в вашей программе другие USB-устройства, достаточно проинициализировать `libusb` один раз — либо с помощью `unc0xx_usb_init()`, либо напрямую. В текущей версии библиотеки для Windows эта функция не делает ничего.

Функция `unc0xx_usb_rescan()`

```
void unc0xx_usb_rescan();
```

Пересканирует USB-шину в поисках подключённых устройств.

Функция `unc0xx_scan()`

```
struct unc0xx_scan_item *unc0xx_scan(int unc0xx_id);
```

Среди устройств, подключённых к USB-шине, находит устройства UNC0xx. Если заданный параметр `unc0xx_id` не равен нулю, находит только устройства с заданным идентификатором (пользовательской частью серийного номера), в противном случае — все устройства типа UNC0xx. Функция возвращает односвязный список, звенья которого представляют собой структуры типа `struct unc0xx_scan_item`. Этот тип описан в заголовочном файле библиотеки следующим образом:

```
struct unc0xx_scan_item {
    int sernum[3];
    int version;
    struct unc0xx_devhdl *dev_hdl;
    struct unc0xx_item *next;
};
```

В поле sernum[2] записывается идентификатор устройства. С каждым обнаруженным устройством устанавливается связь; открытый дескриптор этой связи записывается в поле `dev_hdl`. В поле `version` записывается номер версии прошивки. Если обнаруженное устройство имеет версию, несовместимую с текущей

версией библиотеки, поле dev_hdl получает значение NULL, а поле version *может* содержать 0, если конкретный номер версии прошивки определить не удалось. Для уничтожения списка, закрытия связей с устройствами и освобождения памяти используйте функцию unc0xx_destroy_list (см. ниже).

Функция unc0xx_destroy_list()

```
void unc0xx_destroy_list(struct unc0xx_scan_item *lst);
```

Освобождает память от списка, созданного функцией unc0xx_scan. По умолчанию закрывает каналы связи с устройствами, дескрипторы которых находятся в полях dev_hdl звеньев уничтожаемого списка. Если это нежелательно (например, с некоторыми устройствами вы намерены продолжить работу), присвойте полю dev_hdl соответствующих звеньев списка значение NULL. В этом случае канал связи с устройством необходимо будет позже закрыть самостоятельно с помощью функции unc0xx_close.

Функция unc0xx_rs485_open()

```
struct unc0xx_devhdl* unc0xx_rs485_open(const char *port, int baud,
                                        int id);
```

Открывает соединение с устройством, подключенным к компьютеру через интерфейс RS485 с использованием последовательного порта с именем port. Параметр baud задаёт скорость обмена (бод); указание значения 0 означает использование скорости 19200. Учтите, что если данный конкретный порт уже открыт для работы с другим устройством UNC0xx, его скорость изменена не будет; параметр baud в этом случае игнорируется. Следует отметить, что функция не проверяет наличие устройства; её вызов окончится успешно, если ей удалось открыть файл последовательного порта. В случае, если порт открыть не удалось, возвращается нулевой указатель.

Функция unc0xx_rs485_probe()

```
struct unc0xx_devhdl* unc0xx_rs485_probe(const char *port, int baud,
                                         int id, int *vers);
```

Открывает соединение с устройством, подключенным к компьютеру через интерфейс RS485 (см. описание предыдущей функции) и проверяет наличие устройства с заданным идентификатором, запросив его конфигурацию. В случае успеха в переменную *vers заносится номер версии устройства, а сама функция

возвращает дескриптор открытого соединения. В случае неудачи (в том числе если устройство с заданным идентификатором на запрос не откликнулось) возвращается нулевой указатель.

Функция `unc0xx_close()`

```
void unc0xx_close(struct unc0xx_devhdl *dev_hdl);
```

Закрывает канал связи с устройством. Параметр `dev_hdl` здесь и далее задаёт открытый дескриптор связи с устройством.

Функция `unc0xx_get_fullstate()`

```
int unc0xx_get_fullstate(struct unc0xx_devhdl *dev_hdl,
                         struct unc0xx_global_state_str *data);
```

Позволяет получить содержание основного состояния устройства в виде одной структуры данных, которая соответствует структуре данных в памяти самого устройства. Структура `unc0xx_global_state_str` описана в файле `unc_data.h` и содержит регистр статуса входных и выходных линий, текущие значения счётчиков, связанных с входными линиями, метку времени (количество циклов с момента включения устройства), а также буфер текстовых сообщений.

Функция `unc0xx_get_extrastate()`

```
int unc0xx_get_extrastate(struct unc0xx_devhdl *dev_hdl,
                          struct unc0xx_extra_state_str_common *data);
```

Позволяет получить содержание расширенного состояния устройства. Версии прошивки по 2120131 включительно не поддерживали расширенное состояние. Версии до 2131003 включительно использовали для расширенного состояния структуру, описанную в нынешней версии библиотеки под именем `unc0xx_extra_state_str_legacy`, причём байт `data->page_version` содержал значение `0`. Прошивки версий с 3131008 по 3140909 использовали структуру `unc0xx_extra_state_str_onewire`, в которой байт `page_version` содержит число `1`. Байт `page_seq` во всех перечисленных случаях содержал число `0`.

Начиная с версии 4141014, устройства используют две страницы расширенного состояния, организованные в виде структур `unc0xx_extra_state_str_onewire`, в которой хранится информация о ведомых устройствах 1-Wire, и `unc0xx_extra_state_str_delays`, которая содержит

информацию о текущем наборе отложенных действий. В будущем возможно дальнейшее расширение набора таких структур и увеличение их количества.

Для получения всей информации о расширенном состоянии нужно обратиться к функции `unc0xx_get_extrastate` несколько раз подряд. Для управления этим процессом используется байт `data->page_seq`, младшие семь бит которого обозначают порядковый номер страницы расширенного состояния, а старший бит обозначает, существуют ли страницы с бóльшими номерами. В более старых версиях прошивки этот байт всегда был равен `0`, что означало выдачу страницы с номером `0` и указание, что больше страниц нет.

Правильно организованное управляющее приложение должно предусматривать цикл обращений к функции, оканчивающийся, когда значение выражения `data->page_seq&0x80` после очередного обращения к функции окажется нулевым, и обязательно после каждого обращения анализировать значение байта `page_version`, не делая априорных предположений о том, какова должна оказаться версия очередной страницы расширенного состояния. Структура `unc0xx_extra_state_str_common` содержит поля, одинаковые для всех возможных структур расширенного состояния.

Запрос основного состояния устройства (например, с помощью функции `unc0xx_get_fullstate()`) сбрасывает счётчик страниц расширенного состояния; это можно использовать, чтобы всегда начинать запросы расширенных страниц с номера `0`.

Структуры `unc0xx_extra_state_str_*` описаны в файле `unc_data.h`. Там же описаны константы `uncxstate_page_legacy`, `uncxstate_page_onewire`, `uncxstate_page_delays`, соответствующие значению байта `page_version` для разных версий структуры.

Функция `unc0xx_get_configuration()`

```
int unc0xx_get_configuration(struct unc0xx_devhdl *dev_hdl,
                      struct unc0xx_global_configuration_str *data);
```

Позволяет получить информацию о версии и текущей конфигурации устройства в виде одной структуры данных, которая соответствует структуре данных в памяти самого устройства. Структура `unc0xx_global_configuration_str` описана в файле `unc_data.h` и содержит основные настройки устройства, в том числе номер версии прошивки, регистр аппаратной конфигурации, регистр пользовательской конфигурации, а также регистры реакций, связанные с входными линиями.

Функция `unc0xx_save_to_eeprom()`

```
int unc0xx_save_to_eeprom(struct unc0xx_devhdl *dev_hdl);
```

Передаёт устройству команду на сохранение текущих настроек в EEPROM (действие, выполняемое командой `uncctl -W`).

Функция `unc0xx_restore_from_eeprom()`

```
int unc0xx_restore_from_eeprom(struct unc0xx_devhdl *dev_hdl);
```

Передаёт устройству команду на восстановление настроек из EEPROM в качестве текущих (действие, выполняемое командой `uncctl -E`).

Функция `unc0xx_restore_factory()`

```
int unc0xx_restore_factory(struct unc0xx_devhdl *dev_hdl);
```

Передаёт устройству команду на восстановление заводских настроек в качестве текущих (действие, выполняемое командой `uncctl -F`).

Функция `unc0xx_set_configuration()`

```
int unc0xx_set_configuration(struct unc0xx_devhdl *dev_hdl,
                             unsigned int conf);
```

Позволяет установить новое значение *регистра пользовательской конфигурации* (действие, выполняемое командой `uncctl -G`).

Функция `unc0xx_set_hardware_conf()`

```
int unc0xx_set_hardware_conf(struct unc0xx_devhdl *dev_hdl,
                             unsigned int hwconf);
```

Позволяет установить новое значение *регистра аппаратной конфигурации* (действие, выполняемое командой `uncctl -H`). **Не используйте эту функцию, если только вы не уверены в том, что делаете!** Неправильное применение этой функции, в частности, задание слишком малой длительности внутреннего цикла устройства (два старших байта `hwconf`) может лишить устройство способности обрабатывать запросы по USB; для восстановления работоспособности устройства в этом случае потребуется программатор.

Функция unc0xx_write_e_reg()

```
int unc0xx_write_e_reg(struct unc0xx_devhdl *dev_hdl,
                       int regn, char data[8]);
```

Позволяет записать информацию в один из регистров E00–E47 (действие, выполняемое командой `uncctl -Y`).

Функция unc0xx_clear_msg()

```
int unc0xx_clear_msg(struct unc0xx_devhdl *dev_hdl);
```

Передаёт устройству команду на очистку кольцевого буфера текстовых сообщений (действие, выполняемое командой `uncctl -K`).

Функция unc0xx_reset_1wire()

```
int unc0xx_reset_1wire(struct unc0xx_devhdl *dev_hdl);
```

Передаёт устройству команду на сброс всей информации, связанной с шиной 1-Wire, и переинициализацию подсистемы 1-Wire (действие, выполняемое командой `uncctl -J`).

Функция unc0xx_set_new_id()

```
int unc0xx_set_new_id(struct unc0xx_devhdl *dev_hdl, int id);
```

Изменяет идентификатор устройства (действие, выполняемое командой `uncctl -T`).

Функция unc0xx_perform_action()

```
int unc0xx_perform_action(struct unc0xx_devhdl *dev_hdl, int action);
```

Выполняет операцию, заданную кодом действия (см. § 4.4, стр. 17), переданным через параметр `action`. (действие, выполняемое командой `uncctl -A`).

Функция unc0xx_delayed_action()

```
int unc0xx_delayed_action(struct unc0xx_devhdl *dev_hdl,
                          int action, int delay, int location);
```

Выполняет операцию, заданную кодом действия, переданным через параметр `action`, с задержкой, заданной параметром `delay` (действие, выполняемое командой `uncctl -A` с опцией `-d`). В качестве единицы измерения времени используется *цикл* (приблизительно 0,01 с.). Если параметр `location` равен нулю, отложенное действие размещается в первой свободной строке таблицы отложенных действий; значения `0x10`, `0x11`, ..., `0x19` задают номер строки в таблице (соответственно 0, 1, ..., 9), в которой и размещается отложенное действие; если в заданной строке уже было размещено отложенное действие, оно замещается новым.

Функция unc0xx_set_reactions()

```
int unc0xx_set_reactions(struct unc0xx_devhdl *dev_hdl,
                         int line, unsigned long reaction);
```

Присваивает регистру реакции с номером, заданным параметром `line` (от 0 до 7) значение, заданное параметром `reaction` (действие, выполняемое командой `uncctl -R`).

Функция unc0xx_get_counter_range()

```
int unc0xx_get_counter_range(struct unc0xx_devhdl *dev_hdl,
                             int *min, int *max);
```

Позволяет узнать, сколько входных линий поддерживает данное устройство и каковы номера этих линий. В переменную `min` записывается минимальный номер входной линии, в переменную `max` — максимальный. Для устройств UNC001 эти номера всегда равны, соответственно, `0` и `3`, поскольку устройство поддерживает четыре входные линии с нулевой по третью. Для устройств серии UNC01x в настоящее время возможны пары значений (`0`, `7`), (`0`, `3`), (`4`, `7`) и (`4`, `3`); последняя комбинация показывает, что устройство не имеет входных линий. Комбинация (`4`, `7`) не встречается для устройств, модули которых установлены производителем, поскольку модуль UNC010/in, если он один, устанавливается всегда в первый слот.

Функция `unc0xx_zero_query_counters()`

```
int unc0xx_zero_query_counters(struct unc0xx_devhdl *dev_hdl,
                               int counters[], int ncounters);
```

Считывает и обнуляет счётчики, связанные с входными линиями. Значения счётчиков записываются в массив `counters`. Параметр `ncounters` указывает длину массива. В элемент `counters[0]` записывается значение счётчика для линии с минимальным для данного устройства номером (см. описание функции `unc0xx_get_counter_range()`), в следующий элемент — значение счётчика следующей линии, и т. д. Всего записывается не более чем `ncounters` значений, но и не более, чем количество поддерживаемых устройством линий. Функция возвращает количество записанных значений (для устройств UNC001 это число 4, если только `ncounters` не задан меньше), либо `-1` в случае ошибки.

Функция `unc0xx_dump_eeprom()`

```
int unc0xx_dump_eeprom(struct unc0xx_devhdl *dev_hdl, void* buf);
```

Позволяет прочитать всё содержимое EEPROM. Чтение выполняется блоками по 64 байт; восемь вызова подряд выдадут содержимое всех 512 байт EEPROM. Исполнение команды запроса конфигурации (функции `unc0xx_get_configuration()`) сбрасывает информацию о текущем блоке, так что `dump_eeprom` вновь начинает выдавать блоки с первого. При работе с устройствами, имеющими прошивку старых версий (предшествующих версиям 4xxxxxx) чтение выполнялось блоками по 128 байт в четыре приёма (вместо восьми), в связи с чем параметр `buf` должен указывать на область памяти по размером не менее 128 байт. Установить, с какой версией производится работа, можно по возвращаемому функцией значению, которое равно количеству прочитанных байт (128 или 64), либо -1 в случае ошибки.

Функция `unc0xx_takeover_stream()`

```
int unc0xx_takeover_stream(struct unc0xx_devhdl *dev_hdl);
```

«Захватывает» поток ввода, идущего с устройства, отбирая его у штатного драйвера операционной системы. *В настоящее время не работает под Windows.* Не поддерживается (всегда выдаёт ошибку) для интерфейса RS485.

Функция unc0xx_read_stream()

```
int unc0xx_read_stream(struct unc0xx_devhdl *dev_hdl, char buf[8]);
```

Выполняет очередное чтение порции в 8 байт из потока ввода, идущего с устройства. Поток ввода необходимо предварительно «захватить» функцией `unc0xx_takeover_stream()`. *В настоящее время не работает под Windows.* Не поддерживается (всегда выдаёт ошибку) для интерфейса RS485.

Глобальная переменная unc0xx_usb_timeout

```
extern int unc0xx_usb_timeout;
```

Глобальная переменная, содержит значение USB timeout в миллисекундах (по умолчанию 5000). Не рекомендуется изменять этот параметр, если вы не уверены в своих действиях. В настоящее время при работе под Windows этот параметр игнорируется.

Функция unc0xx_get_errormsg()

```
const char *unc0xx_get_errormsg(struct unc0xx_devhdl *dev_hdl);
```

Возвращает указатель на текстовое сообщение, соответствующее последней произошедшей ошибке при работе с заданным устройством. Может вернуть адрес неизменяемой области памяти, так что пытаться модифицировать возвращённую строку не следует. В некоторых случаях возвращает нулевой указатель, что также следует учитывать в работе.