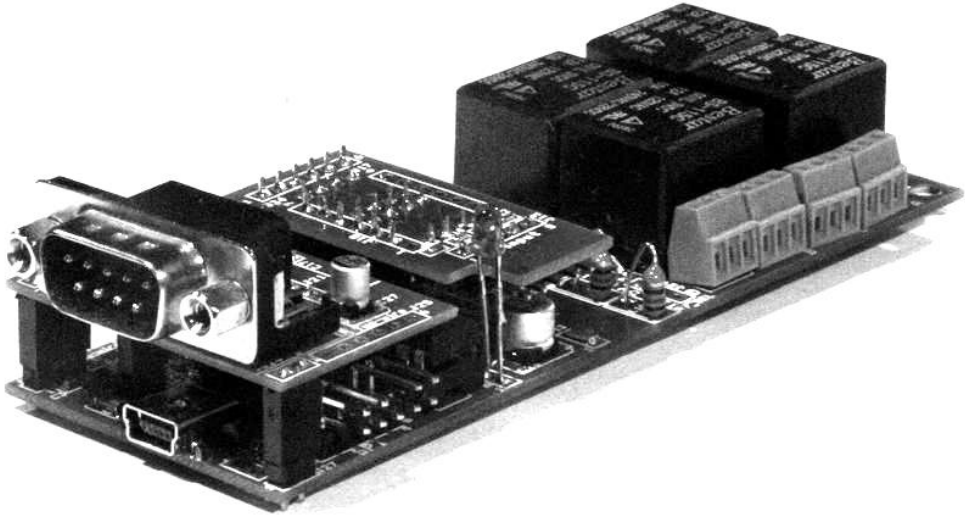
ООО «Юниконтроллерз» Unicontrollers Ltd.

Устройство управления электрическими цепями универсальное четырёхканальное

UNC001

руководство по эксплуатации



Москва – 2013

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

Отдел технической поддержки: support.unc@unicontrollers.com

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

Содержание

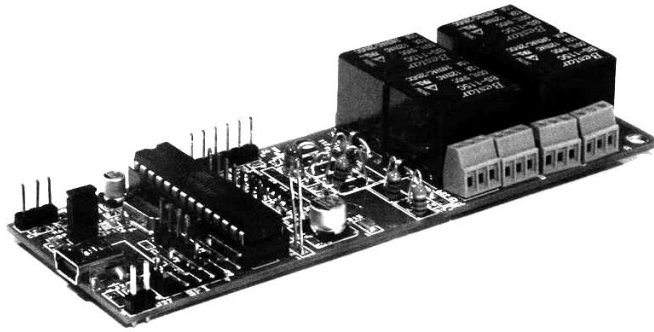


Рис. 1.1: Внешний вид устройства UNC001-1

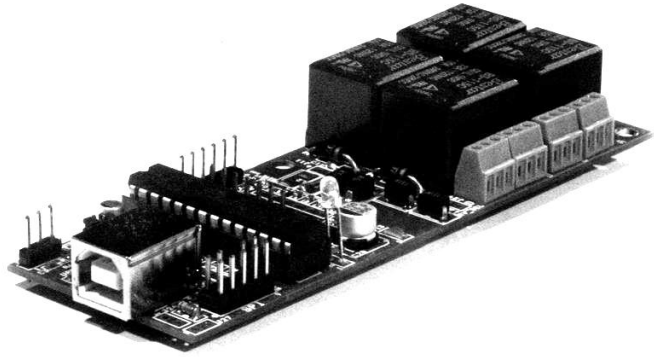


Рис. 1.2: Внешний вид устройства UNC001-2

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).

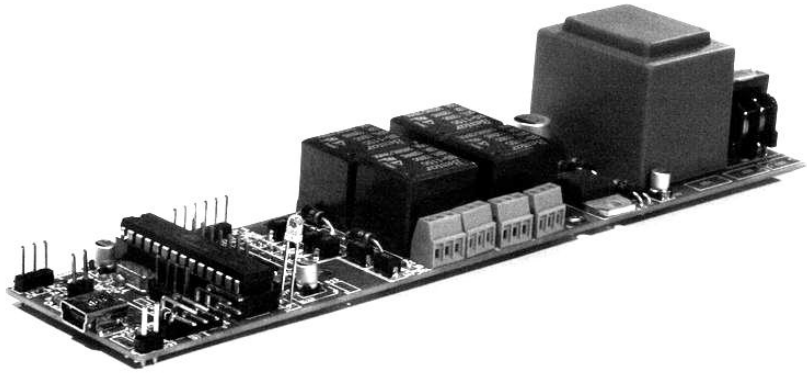


Рис. 1.3: Внешний вид устройства UNC001-3

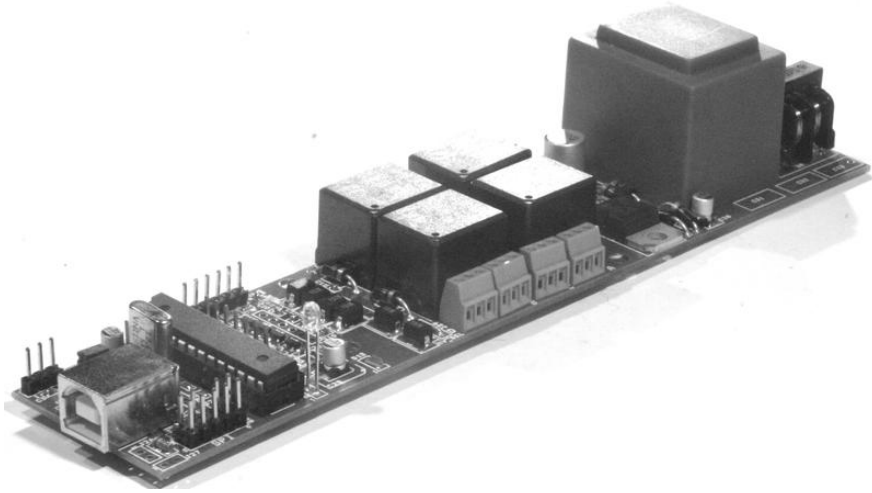


Рис. 1.4: Внешний вид устройства UNC001-4

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

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 компьюте-

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			

ру, оптическим диском, содержащим программное обеспечение и инструкцию по эксплуатации, а также техническим паспортом. Необходимо учитывать, что 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.

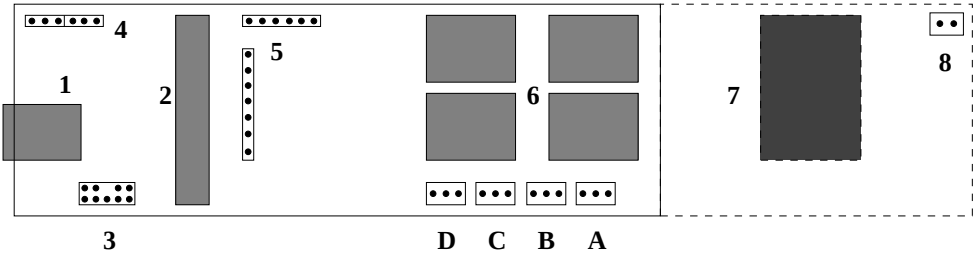
Основные технические характеристики устройств UNC001 приведены в табл. 1.2.

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



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

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 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.

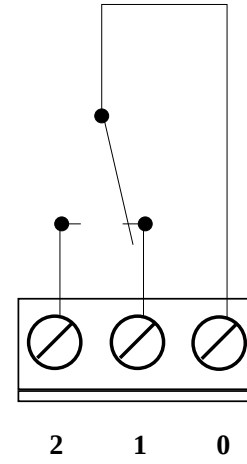


Рис. 3.1: Схема коммутируемой (выходной) цепи

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-

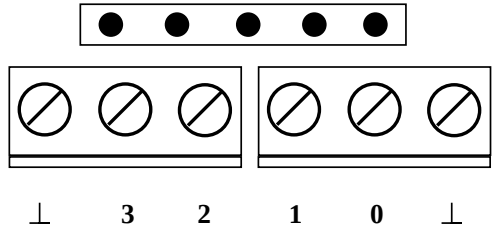


Рис. 3.2: Расположение контактов клеммника для подключения входных цепей

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

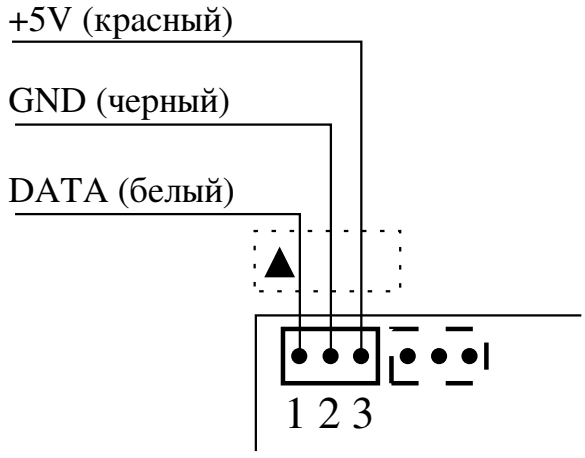


Рис. 3.3: Схема подключения переходника UNC001/1w

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

Семантика кодов действий и значений расширенных регистров, описанная в настоящем руководстве, соответствует версии прошивки 2130915 и более поздним. 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 прошитой 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 настраивается с достаточно высокой погрешностью ($\pm 15\%$), поэтому использовать внутренний таймер устройства для измерения реального времени (например, для включения или выключения цепей в определённое время суток) не представляется возможным; с этой целью необходимо использовать таймер управляющего 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 вы планируете использовать больше одного устройства с одним компьютером,

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

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

Таблица 4.1: Регистр аппаратной конфигурации

рекомендуется сразу же сменить заводское значение на другое, например 0001, 0002 и т. д. (см. стр. 22).

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. Биты с 15 по 8 не используются. Младший байт регистра аппаратной конфигурации при использовании прошивки на устройствах UNC001-x **должен иметь значение** 0x04, в противном случае возможна некорректная работа устройства вплоть до его физического повреждения. **Изменение содержимого регистра аппаратной конфигурации средствами поставляемого программного обеспечения возможно, но настоятельно не рекомендуется; компания-производитель не несёт ответственности за возможные последствия.**

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.

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

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 обнаруженных датчиков DS18x20?
19	0x00080000	Сканировать расширенные регистры Exx на предмет идентификаторов «таблеток» iButton?
18	0x00040000	Сканировать расширенные регистры Exx на предмет событий, связанных с температурой?
17-9	0x0003fe00	зарезервированы
8	0x00000100	генерировать поток информации на USB?
7-0	0x000000ff	антидребезг: количество циклов, после которого входная линия считается сменившей статус

Таблица 4.2: Регистр пользовательской конфигурации

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 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.

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

Таблица 4.3: Регистр реакции

Остальные биты регистра реакций (биты 31, 25 и 24) зарезервированы для будущих версий; в настоящее время они не используются.

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;
- значения 0x70 и 0x7e означают, соответственно, главный и подчинённые регистры для взаимодействия с температурным датчиком;
- 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. 7.

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.

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

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

Таблица 4.4: Регистр состояния

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

Таблица 4.5: Регистр программного состояния (fsm-регистр)

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 § 7.2.

Регистр таймера содержит количество *циклов работы*, прошедших с момента включения питания. Цикл **приблизительно** соответствует 0,01 секунды, однако погрешности могут достигать довольно значительных величин.

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 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

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	
0x0a	(assign) изменение FSM-регистра	старшие 4 бита — номер FSM-регистра (от 0 до 4); 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; 0..47 — отменить отложенное обращение к E-регистру 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

Таблица 4.6: Старший полубайт кода действия

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 дей-

ствия 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` (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 `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 § 7.2.

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 задаёт 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. Подробное описание этих возможностей вы найдёте в §§ 7.1 и 7.2.

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.

V. Программное обеспечение для персонального компьютера

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

1. утилита командной строки `uncctl`;
2. программа UNC Monitor с графическим интерфейсом пользователя;
3. библиотека управляющих функций `uncusb` 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. 21). Библиотека `uncusb` (см. стр. 39) 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) и сборка из исходных текстов.

Для установки готовых исполняемых файлов скачайте их (это файлы `uncctl` и `uncmonitor`) с сайта производителя устройства, либо, при наличии в комплекте устройства оптического диска, возьмите эти файлы из корневого каталога на диске и скопируйте их в директорию `/usr/local/bin` в вашей системе (это потребует полномочий администратора — пользователя `root`). Программы сразу же готовы к работе.

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 программы `uncctl`, библиотеки `uncusb` и прошивки находятся в файле `unc0xxx-NNNNNNN.tgz`, где `NNNNNNN` — номер версии; 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` и `libuncusb.a`), будет собрано.

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 `libuncusb.a` в директорию `/usr/local/lib`, а файл `uncusb.h` — в директорию `/usr/local/include`.

Установка программного обеспечения в операционных системах семейства Windows сводится к копированию файла `uncctl.exe` в любую директорию, доступную через `PATH` (или любую другую, но тогда придётся для вызова `uncctl` 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.

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`.

VI. Управление устройством с помощью программы `uncctl`

Программа `uncctl` (в системах семейства Windows — `uncctl.exe`) представляет собой утилиту командной строки, обеспечивающую доступ ко всем функциям устройства UNC001. 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`).

Команда `uncctl -L` выдаёт список устройств UNC001 и UNC01x, подключённых к данному компьютеру. 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]   2130915
3333  [0002:0043:3333]         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. 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`.

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, а для температурных датчиков — считанная с них температура. 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), 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 -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.

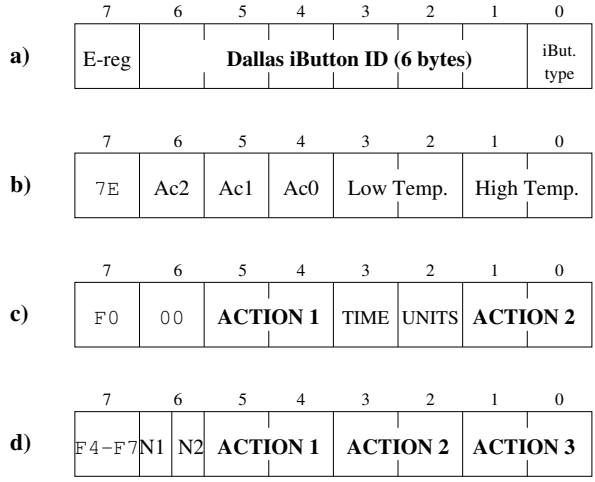


Рис. 7.1: Структура E-регистров при различных способах использования

VII. 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. 7.1.

7.1. Сложные и «отложенные» реакции на события

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.

Для задания сложного действия подходит любой из расширенных регистров; старший байт (байт № 7) должен содержать значение `0xf0`, чтобы показать, что данный регистр используется для хранения сложного действия. Байт № 6 для за-

Старший байт	Способ использования регистра
00–47	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. 7.1 (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 предыдущему случаю (рис. 7.1 (a)), отличается только значением старшего байта.
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. 7.1 (b).
7F	Регистр не используется, но в следующих за ним могут находиться идентификаторы 1-Wire-устройств.
F0	Регистр содержит описание «сложного действия». Структура регистра показана на рис. 7.1 (c).
F4–F7	Регистр содержит описание «условного действия». Структура регистра показана на рис. 7.1 (d).
FF	Регистр не используется.

Таблица 7.1: Возможные значения старшего байта E-регистра

дания сложного действия не используется и должен быть выставлен в значение 0x00.

Сложное действие 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.

Устройство способно в каждый момент времени «помнить» четыре «отложенных действия» (каждое со своей задержкой и со своим кодом действия). Попытка установить больше отложенных действий ни к чему не приведёт (соответствующее действие будет просто проигнорировано).

Рассмотрим пример. Занесём в регистр `E19` ($19_{10} = 13_{16}$) сложное действие, 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.

7.2. 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 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.

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

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 § 7.6). 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 ($2F_{16} = 47_{10}$).

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 42 'F467 0E29 0F00 0F00'  # 2A
uncctl -E 41 'F000 0A31 0201 0E28'  # 29
uncctl -E 40 'F601 0A30 0A32 0A30'  # 28
uncctl -E 39 'F612 0A30 0A33 0A30'  # 27
uncctl -E 38 'F623 0A30 0A34 0A30'  # 26
uncctl -E 37 'F000 0E24 0100 0E23'  # 25
uncctl -E 36 'F634 0F00 0E2F 0F00'  # 24
uncctl -E 35 'F000 0A30 0000 0F00'  # 23
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.

7.3. 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 дважды в секунду) будет выполняться сканирование шины 1-Wire на предмет подключённых устройств.

Прошивка способна работать не более чем с шестью 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 1-Wire-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 § 7.4.

18-й бит регистра пользовательской конфигурации включает обработку верхних и нижних границ температурных диапазонов для датчиков температуры. Подробности приведены ниже в § 7.5.

Например, команда

```
uncctl -G 00fc0001
```

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

7.4. 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 шестнадцатиричное 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 46 f000000103010002
```

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 1-Wire-устройства, 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`).

7.5. 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 $\frac{1}{16}{}^oC$ — например, `0xfff8` соответствует $-0{,}5^oC$, а `0x00f4` — $+15{,}25^oC$.

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^oC, верхнюю — 28^oC (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 § 7.2.

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.

7.6. 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"
```

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

8.1. Общее описание

В состав программного обеспечения для ПК, обеспечивающего работу с устройствами UNC001 и UNC01x, входит модуль, реализующий набор функций для связи с устройством через интерфейс USB. Исходный текст модуля находится в файле `uncusb.c`; соответствующий заголовочный файл называется `uncusb.h`. При сборке программного обеспечения создаётся статическая библиотека `libuncusb.a`; в текущей Unix-версии она состоит из одного объектного модуля (`uncusb.o`), так что можно собирать программы непосредственно с этим модулем, однако в будущем возможно появление дополнительных модулей, поэтому рекомендуется использовать библиотеку. В версии для Windows в эту библиотеку также включается модуль `hidapi`, так что необходимо либо использовать библиотеку, либо подключать вручную оба модуля.

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

```
#include <uncusb.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 `uncusb`; 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 -luncusb \
    `libusb-config --libs`  -o myprogram
```

Помните, что библиотека `libusb` должна быть подключена **после** библиотеки `uncusb`. 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 «`unc001_`». Библиотека содержит следующие публичные объекты (функции, имеющие тип возвращаемого значения `int`, возвращают `-1` в случае ошибки, если в описании не указано иное).

[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.

8.2. Принципы взаимодействия с библиотекой uncusb

Перед началом работы необходимо инициализировать библиотеку, обеспечивающую работу с USB; это можно сделать вызовом

```
unc001_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 `uncusb` инициализации не требует.

После инициализации следует выполнить сканирование USB-шины в поисках устройств, которыми можно управлять. Это делается с помощью функции `unc001_scan()`, которая получает на вход один параметр — «пользовательский идентификатор» нужного устройства (см. стр. 22), либо число 0 для поиска всех устройств UNC0xx, подключённых к компьютеру в настоящий момент. Поскольку обнаружено может быть больше одного устройства, функция формирует односвязный список, состоящий из структур типа `struct unc001_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 `unc001_destroy_list()`. Например:

```
struct unc001_item *list;
unc001_usb_init();
list = unc001_scan(0);
if(list) {

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

    unc001_destroy_list(list);
} else {
    fprintf(stderr, "No devices found");
}
```

Структура `struct unc001_item` содержит по меньшей мере следующие поля:

```
int                  sernum[3];
int                  version;
void *               dev_hdl;
struct unc001_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 абстрактный 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 unc001_item *list, *t;
int cn = 0;
unc001_usb_init();
list = unc001_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);
unc001_destroy_list(list);
```

Функция `unc001_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 unc001_item *list;
void *dh = NULL;
unc001_usb_init();
list = unc001_scan(0);
if(list) {
    dh = list->dev_hdl;
    list->dev_hdl = NULL;
    unc001_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, окажется закрыто.

Закрыть соединение с отдельным устройством позволяет функция `unc001_close()`:

```
unc001_close(dh);
```

Используя идентификатор открытого соединения, вы можете осуществлять взаимодействие с устройством. В частности, вы можете запросить конфигурацию и состояние устройства с помощью функций `unc001_get_fullstate()`, `unc001_get_extrastate()` и `unc001_get_configuration()`. Заставить устройство выполнить то или иное действие можно при помощи функции `unc001_perform_action()`, и так далее. Подробное описание функций библиотеки `uncusb` приведено в следующем параграфе.

8.3. Справочник по функциям

Функция `unc001_usb_init()`

```
void unc001_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 `unc001_usb_init()`, либо напрямую. В текущей версии библиотеки для Windows эта функция не делает ничего.

Функция `unc001_usb_rescan()`

```
void unc001_usb_rescan();
```

Пересканирует USB-шину в поисках подключённых устройств.

Функция `unc001_scan()`

```
struct unc001_item *unc001_scan(int unc001_id);
```

Среди устройств, подключённых к USB-шине, находит устройства UNC0xx. Если заданный параметр `unc001_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 unc001_item`. Этот тип описан в заголовочном файле библиотеки следующим образом:

```
struct unc001_item {
    int sernum[3];
    int version;
    void *dev_hdl;
    struct unc001_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 `libuncusb`, 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 `unc001_destroy_list` (см. ниже).

Функция `unc001_destroy_list()`

```
void unc001_destroy_list(struct unc001_item *lst);
```

Освобождает память от списка, созданного функцией `unc001_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 `unc001_close`.

Функция `unc001_close()`

```
void unc001_close(void *dev_hdl);
```

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

Функция `unc001_get_fullstate()`

```
int unc001_get_fullstate(void *dev_hdl,
                         struct unc001_global_state_str *data);
```

Позволяет получить содержание основного состояния устройства в виде одной структуры данных, которая соответствует структуре данных в памяти самого устройства. Структура `unc001_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 `unc001_get_extrastate()`

```
int unc001_get_extrastate(void *dev_hdl,
                          struct unc001_extra_state_str_common *data);
```

Позволяет получить содержание расширенного состояния устройства. Версии прошивки до 2131003 включительно использовали для расширенного состояния структуру `unc001_extra_state_str_0`, причём байт `data->page_version` содержал значение `0`. Прошивки версий 3131008 и новее используют структуру `unc001_extra_state_str_1`, байт `page_version` содержит число `1`. В будущем возможно дальнейшее расширение набора возможных структур и одновременное наличие в устройстве более чем одной структуры; в этом случае для получения всей информации о расширенном состоянии нужно будет обратиться к функции `unc001_get_extrastate` несколько раз подряд. Для управления этим процессом предусмотрен байт `data->page_seq`, младшие семь бит которого будут обозначать порядковый номер страницы расширенного состояния, а старший бит — обозначать, существуют ли страницы с большими номерами. В нынешней версии этот байт всегда равен `0`, то есть выдаётся страница с номером `0` и указывается, 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.

Структуры `unc001_extra_state_str_*` описаны в файле `unc_data.h`.

Функция `unc001_get_configuration()`

```
int unc001_get_configuration(void *dev_hdl,
                             struct unc001_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 `unc001_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 `unc001_save_to_eeprom()`

```
int unc001_save_to_eeprom(void *dev_hdl);
```

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

Функция `unc001_restore_from_eeprom()`

```
int unc001_restore_from_eeprom(void *dev_hdl);
```

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

Функция `unc001_restore_factory()`

```
int unc001_restore_factory(void *dev_hdl);
```

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

Функция `unc001_set_configuration()`

```
int unc001_set_configuration(void *dev_hdl, unsigned int conf);
```

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

Функция `unc001_set_hardware_conf()`

```
int unc001_set_hardware_conf(void *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!** Компания-производитель снимает с себя всякую ответственность за возможные последствия.

Функция `unc001_write_e_reg()`

```
int unc001_write_e_reg(void *dev_hdl, int regn, char data[8]);
```

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

Функция `unc001_clear_msg()`

```
int unc001_clear_msg(void *dev_hdl);
```

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

Функция `unc001_set_new_id()`

```
int unc001_set_new_id(void *dev_hdl, int id);
```

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

Функция `unc001_perform_action()`

```
int unc001_perform_action(void *dev_hdl, int action);
```

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

Функция `unc001_set_reactions()`

```
int unc001_set_reactions(void *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 `unc001_get_counter_range()`

```
int unc001_get_counter_range(void *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.

Функция `unc001_zero_query_counters()`

```
int unc001_zero_query_counters(void *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 `unc001_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 `unc001_dump_eeprom()`

```
int unc001_dump_eeprom(void *dev_hdl, void* buf);
```

Позволяет прочитать всё содержимое EEPROM. Чтение выполняется блоками по 128 байт; четыре вызова подряд выдадут содержимое всех 512 байт EEPROM. Исполнение команды запроса конфигурации (функции `unc001_get_configuration()`) сбрасывает информацию о текущем блоке, так что `dump_eeprom` вновь начинает выдавать блоки с первого. Параметр `buf` должен указывать на область памяти по размером не менее 128 байт.

Функция `unc001_takeover_stream()`

```
int unc001_takeover_stream(void *dev_hdl);
```

«Захватывает» поток ввода, идущего с устройства, отбирая его у штатного драйвера операционной системы. *В настоящее время не работает под Windows.*

Функция `unc001_read_stream()`

```
int unc001_read_stream(void *dev_hdl, char buf[8]);
```

Выполняет очередное чтение порции в 8 байт из потока ввода, идущего с устройства. Поток ввода необходимо предварительно «захватить» функцией `unc001_takeover_stream()`. *В настоящее время не работает под Windows.*

Глобальная переменная `unc001_usb_timeout`

```
extern int unc001_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 `unc001_strerror()`

```
const char *unc001_strerror();
```

Возвращает указатель на текстовое сообщение, соответствующее последней произошедшей ошибке. 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.