1 Disclaimer

```
#setencoding=utf8
#
# Copyright (C) 2012 Sergey Metelev
#
# version: v0.05-1-gaacc511
#

# The following terms apply to usbcc.py and usbccpy.pdf files
#
# The author hereby grant permission to use, copy, modify,
# distribute, and license this software and its documentation
# for any purpose, provided that existing copyright notices
# are retained in all copies and that this notice is included
# verbatim in any distributions. No written agreement,
# license, or royalty fee is required for any of the
# authorized uses.  Modifications to this software may be
# copyrighted by their authors and need not follow the
# licensing terms described here, provided that the new terms
# are clearly indicated on the first page of each file where
# they apply.
#
# IN NO EVENT SHALL THE AUTHOR OR DISTRIBUTORS BE LIABLE TO
# ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR
# CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS
# SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF,
# EVEN IF THE AUTHOR HAVE BEEN ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
#
# THE AUTHOR AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY
# WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
# WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
# PURPOSE, AND NON-INFRINGEMENT.  THIS SOFTWARE IS PROVIDED
# ON AN "AS IS" BASIS, AND THE AUTHOR AND DISTRIBUTORS HAVE
# NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
# ENHANCEMENTS, OR MODIFICATIONS.
```

Адаптер usb-cc217.10, работа под ОС Linux

С.В. Метелёв, email: metelev.sv@yandex.ru

Содержание

2 Введение

Адаптер предназначен для подключения контроллера КАМАК 217.10 к шине USB. Адаптер создан в Лаборатории РадиоЭлектроники (ЛРЭ) Отделения Физики Высоких Энергий (ОФВЭ) Петербургского института ядерной физики (ПИЯФ). Домашняя страница проекта находится по адресу http://dbserv.pnpi.spb.ru/hepd/red/products/Adapter.html

В данном описании изложен опыт работы с адаптером в Linux, с использованием интерпретатора Python. Приводится исходный текст программного модуля usbcc.py, осуществляющего работу с адаптером.

Выбор операционной системы и языка[1] обусловлен тем, что программный модуль используется как часть программы управления рентгеновской установкой. В то же время работа может быть полезна в более широком круге случаев: для интерактивной работы с КАМАК, в том числе в среде Windows, а так же как прототип для написания программы на C. Популярность и постоянное развитие интерпретатора Python даёт некоторые гарантии того, что работа не будет ограничена рамками его возможностей.

В Linux для работы с usb-устройствами используется библиотека libusb. У неё есть две ветви разработки, отличающиеся номерами версий: 0.1 и 1.0, но с несовместимыми API. Ещё есть отделившаяся от libusb 1.0 библиотека OpenUSB. В Windows кроме библиотеки libusb можно использовать собственный драйвер фирмы Cypress, чья микросхема CY7C68013 была использована как основа адаптера. В Linux такой возможности нет (Cypress говорит об этом).

Все клоны libusb могут быть использованы совместно с python библиотекой pyusb.

Следует упомянуть что для загрузки прошивки в адаптер под Linux используется Open Source программа fxload, а сам процесс загрузки интегрируется в ОС. При переносе python-подхода в Windows придётся отдельно продумывать способ загрузки прошивки.

3 Инсталляция

В состав архива входят

- `FX2_to_CC217.hex` файл прошивки
- `11-usb-cc217.10.rules` файл-правило для автоматической загрузки прошивки при помощи udev и замены прав доступа к устройству

[1] Использовались: Python 2.7, python-библиотека pyusb 1.0.0_alpha1, C-библиотека libusb 1.0.8 как backend. Дистрибутив Gentoo, ядро 3.0.6

- `usbccpy.pdf` подробное описание (этот файл)
- usbcc.py программный модуль python

Кроме того для работы нужны библиотека libusb, менеджер устройств udev и интерпретатор Python , которые как правило входят в состав системы Linux. А так же модуль pyusb и программа fxload для загрузки прошивки.

Как правило производители Linux дистрибутивов поддерживают так называемые репозитарии, в которых собраны версии всех доступных Open Source программ в виде пакетов предназначенных именно для данного дистрибутива, уже откомпилированных и доступных для установки. В состав дистрибутива входит набор команд, который позволяет управлять установкой и удалением пакетов и обновлять список доступных пакетов.

В Gentoo необходимые пакеты могут быть установлены с помощью команды (необходимы права суперпользователя) `emerge pyusb fxload`

Затем нужно поместить файл FX2_to_CC217.hex в директорию /usr/share/usb и присвоить ему права доступа 644, владельца и группу root. Затем файл 11-usb-cc217.10.rules в директорию /etc/udev/rules.d с теми же правами. Для этого необходимо выполнить команды (от имени суперпользователя):

```
cp FX2_to_CC217.hex /usr/share/usb
chmod 644 /usr/share/usb/FX2_to_CC217.hex
chown root:root /usr/share/usb/FX2_to_CC217.hex
cp 11-usb-cc217.10.rules /etc/udev/rules.d
chmod 644 /etc/udev/rules.d/11-usb-cc217.10.rules
chown root:root /etc/udev/rules.d/11-usb-cc217.10.rules
```

Теперь при подключении устройства в него будет загружена прошивка, и оно появится как новое устройство, причем права доступа будут открытыми на чтение и запись для всех пользователей.

4 Интерактивная работа

В состав дистрибутива Python входит пакет idle, который представляет собой среду для запуска python-программ. Для запуска интерактивной сессии с возможностью использования команд модуля usbcc.py нужно перейти в директорию с программой запустить модуль следующим образом:

```
idle -r usbcc.py
```

При интерактивном запуске модуль подключает лог-систему с выводом сообщений на экран, посредством которой и взаимодействует с пользователем. Пример сессии, запущенной таким образом:

```
Python 2.7.2 (default, Nov 23 2011, 12:24:37)
[GCC 4.4.5] on linux2
Type "copyright", "credits" or "license()" for more information.
>>>
INFO  10:21:57 Найден инициализированный адаптер usb-cc217.10
DEBUG 10:21:57 чтение  -1, A,  F,  N,  L,LAM,  X,  Q,  S,  данные
DEBUG 10:21:57             0,  0,  0,  1, 31,  0,  0,  1, 0
DEBUG 10:21:57 чтение  -1, A,  F,  N,  L,LAM,  X,  Q,  S,  данные
DEBUG 10:21:57             0,  0, 22,  1, 31,  0,  0,  1, 0
DEBUG 10:21:57 загружается 3 команд
DEBUG 10:21:57 чтение   0, A,  F,  N,  L,LAM,  X,  Q,  S,  данные
DEBUG 10:21:57             3, 25,  8,  1, 31,  1,  0,  1, 0
DEBUG 10:21:57            12,  1,  8,  1, 31,  1,  1,  1, 0
DEBUG 10:21:57             4,  0,  8,  1, 31,  1,  1,  1, 493567
DEBUG 10:21:57 чтение   1, A,  F,  N,  L,LAM,  X,  Q,  S,  данные
DEBUG 10:21:57             3, 25,  8,  1, 31,  1,  0,  1, 0
DEBUG 10:21:57            12,  1,  8,  1, 31,  1,  1,  1, 0
DEBUG 10:21:57             4,  0,  8,  1, 31,  1,  1,  1, 493567
DEBUG 10:21:57 чтение   2, A,  F,  N,  L,LAM,  X,  Q,  S,  данные
DEBUG 10:21:57             3, 25,  8,  1, 31,  1,  0,  1, 0
DEBUG 10:21:57            12,  1,  8,  1, 31,  1,  1,  1, 0
DEBUG 10:21:57             4,  0,  8,  1, 31,  1,  1,  1, 493567
INFO  10:21:57 Загружен модуль usbcc.py
>>>
```

Вывод сообщений осуществляется лог-обработчиками, каждый из которых может иметь свой уровень подробности и устройство (файл) вывода. При тестовых запусках работает один обработчик, который выводит сообщения на экран, сообщения важности до DEBUG уровня, то есть почти все.

Вывод представляет собой список возвращаемых значений под шапкой, которая напоминает об их смысле. За справками нужно обращаться к описанию КАМАК и функций decode() и encode().

Вначале выполнилось две команды, указанные в разделе Команды, выполняемые при автономном запуске:

```
decode(Camac(0,0,0,0))
decode(Camac(22,0,0,1))
```

Команды в данном случае служат только для тестирования. Тестирование заключается в том, что при отключенном крейте переменные L, LAM, S содержат нули.

Начиная со строчки “загружается 3 команд” в адаптер загружается и запускается на выполнение последовательность из 3 КАМАК команд. Результаты выполнения этих команд помещаются поочерёдно в два буфера, так что пока один буфер читается компьютером, другой записывается результатами работы последовательности КАМАК команд. При заполнении обеих буферов выполнение команд приостанавливается. Кроме того, выполнение последовательности можно приостановить и потом продолжить снова.

Результат работы последовательности читается 3 раза (команда range(3)), но там может стоять сколь угодно большое число.

На настоящий момент после инициализации адаптера последовательность команд можно в него загрузить только один раз, при попытке повторной загрузки произойдёт аварийный выход по таймауту и будет выдано такое сообщение:

```
ERROR 10:22:48 Произошло исключение! причина: Operation timed out
ERROR 10:22:48 Для повторной загрузки последовательности
ERROR 10:22:48 отключите и вновь подключите адаптер к шине USB!

Traceback (most recent call last):
  File "usbcc.py", line 320, in <module>
    s=sequencer(sequence)
  File "usbcc.py", line 206, in __init__
    self.write(sequence)
  File "usbcc.py", line 233, in write
    sys.exit()
SystemExit
>>>
```

После выполнения запрограммированной последовательности idle стандартное python-приглашение ввести новую команду: >>>. Доступны как встроенные команды Python, так и те, которые определены в модуле usbcc.py Например, можно ещё раз выполнить команду *Camac(0,0,0,0)*, без обработки функцией decode():

```
>>> Camac(0,0,0,0)
array('B', [0, 0, 0, 252, 0, 0, 0, 253])
>>>
```

Модуль можно использовать как библиотеку. Для этого нужно в основной программе включить строку `import usbcc`. После этого можно обращаться к именам определённым в модуле используя запись `usbcc.Camac()`

5 Подробное описание подключения

Этот раздел предназначен для случая, когда что-то идёт не так. В случае если адаптер заработал, читать его не обязательно. Подключение устройства двухэтапное. На первом этапе в списке usb-устройств появляется устройство, которое позволяет загрузить прошивку. Список всех usb-устройств в системе можно посмотреть командой **lsusb** (см. Рис. 1). После загрузки прошивки устройство переподключается и в списке появляется новое устройство, которое осуществляет функции адаптера (см. Рис. 2)

```
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 003: ID 1a40:0101 TERMINUS TECHNOLOGY INC. USB-2.0 4-Port HUB
Bus 002 Device 002: ID 0566:3002 Monterey International Corp.
Bus 001 Device 026: ID 04b4:8613 Cypress Semiconductor Corp. CY7C68013 EZ-USB FX2 USB 2.0
Development Kit
Bus 001 Device 020: ID 1e4e:0102
Bus 001 Device 005: ID 2001:f103 D-Link Corp. DUB-H7 7-port USB 2.0 hub
Bus 001 Device 006: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x Composite Device
Bus 001 Device 021: ID 1e4e:0102
Bus 001 Device 022: ID 1e4e:0102
Bus 001 Device 023: ID 1e4e:0102
Bus 001 Device 024: ID 1e4e:0102
```

Рис. 1: Вывод lsusb до загрузки прошивки, в 5-й строчке подключенный адаптер сообщает о себе. В данном случае адрес Bus 001 Device 026.

Для загрузки прошивки необходимо установить программу fxload. Процедура установки зависит от дистрибутива. В Gentoo Linux необходимо выполнить команду **emerge fxload**.

Прошивку можно загрузить вручную. Допустим, файл с прошивкой лежит в директории /usr/share/usb, а устройство расположено по адресу Bus 001, Device 026. Тогда необходимо набрать команду (набирать в одну строку):

```
fxload -v -t fx2 -I /usr/share/usb/FX2_to_CC217.hex
-D /dev/bus/usb/001/026
```

Для того, чтобы прошивка автоматически загружалась при подключении устройства нужно создать файл с таким именем и содержимым:

```
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 003: ID 1a40:0101 TERMINUS TECHNOLOGY INC. USB-2.0 4-Port HUB
Bus 002 Device 002: ID 0566:3002 Monterey International Corp.
Bus 001 Device 028: ID 0547:1002 Anchor Chips, Inc. Python2 WDM Encoder
Bus 001 Device 020: ID 1e4e:0102
Bus 001 Device 005: ID 2001:f103 D-Link Corp. DUB-H7 7-port USB 2.0 hub
Bus 001 Device 006: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x Composite Device
Bus 001 Device 021: ID 1e4e:0102
Bus 001 Device 022: ID 1e4e:0102
Bus 001 Device 023: ID 1e4e:0102
Bus 001 Device 024: ID 1e4e:0102
```

Рис. 2: Вывод lsusb после загрузки прошивки и переподключения. Адрес устройства сменился: Bus 001 Device 028

```
/etc/udev/rules.d/11-usb-cc217.10.rules

ATTRS{idVendor}=="0547", ATTRS{idProduct}=="1002", MODE="666"
SUBSYSTEMS=="usb", ACTION=="add", ATTRS{idVendor}=="04b4",\
    ATTRS{idProduct}=="8613", RUN+="/sbin/fxload -v -t fx2 -I\
    /usr/share/usb/FX2_to_CC217.hex -D $tempnode"
```

Теперь, если всё сделано правильно, при подключении адаптера прошивка должна автоматически в него загрузиться и права доступа должны позволять запускать управление от имени любого пользователя.

6 Usbcc.py – version: v0.05-1-gaacc511

В этом разделе содержится исходный текст программы с подробными комментариями. В конце раздела приводится список команд, который выполняется при автономном запуске модуля. Его можно менять, тем самым модуль можно рассматривать как автономное приложение для запуска простых последовательностей команд КАМАК.

6.1 Общее описание взаимодействия с адаптером

6.1.1 Одиночные команды

Одиночные команды передаются адаптеру с помощью Setup-пакетов, тип передачи CONTROL (см. спецификацию USB [2, 3, 4]). В модуле pyusb эта работа реализована с помощью функции *ctrl_transfer* (). Параметры запроса соответствуют полям пакета Setup. В модуле pyusb функция определяется так:

ctrl_transfer (bmRequestType, bRequest, wValue, wIndex, \
data_or_wLength, timeout) :

Описание аргументов:

bmRequestType определяет направление потока данных, тип запроса и получателя

bRequest определяет запрос

wValue = 0 смысл поля определяется запросом. По умолчанию 0.

wIndex = 0 смысл поля определяется запросом. По умолчанию 0.

data_or_wLength при записи содержит побайтную последовательность передаваемых данных, например: *bytearray* ((0, 1, 2, 3, 4, 5, 6, 7)). В этом случае процедура возвращает количество записанных байт. При чтении содержит количество байт, которые необходимо прочитать, а процедура возвращает прочитанные данные, тип array.

timeout = 100 по умолчанию таймаут 100 миллисекунд.

Поле, bmRequestType конструируется следующим образом [4]:

D7 направление передачи данных в фазе данных; 0 OUT, "от компьютера 1 IN, "в компьютер"

D6, D5 тип требования: 0 стандартное, 1 класса, 2 продавца (производителя), 3 зарезервировано;

D4...D0 получатель данного требования: 0 устройство, 1 интерфейс, 2 точка, 3 остальные, 4...31 зарезервировано

Остальные поля, и в случае необходимости данные для передачи, формируются в соответствии с требованием.

6.1.2 Последовательности команд

Последовательности передаются с помощью передач типа BULK [2, 3]. Функции, осуществляющие чтение и запись, очень похожи друг на друга.
device.*write* (OutEndPt, sequence, Interface, timeout)
device.*read* (InEndPt, length, Interface, timeout)
Оба вызова содержат в качестве параметров EndPt—номер точки доступа, и необязательные Interface—номер интерфейся и timeout—ожидание ответа, в миллисекундах. Команда write дополнительно содержит sequence, байтовую последовательность для записи, и возвращает количество записанных байт. Команда read содержит length—количество байт для чтения и возвращает прочитанные данные. Формат данных определяется прошивкой. Он подробно изложен в аннотациях к функциям encode() и decode()

6.2 Подготовительная часть

6.2.1 Инициализация ведения лог-файла

```
64 import logging
65 log = logging.getLogger('manager.usb')
```

при автономном запуске добавляем обработчик сообщений и устанавливаем уровень для выводимых сообещний DEBUG

```
69 if __name__ == "__main__":
70     import logging.handlers
71     common_format = '%(levelname)-6s%(asctime)s %(message)s'
72     date_format = '%H:%M:%S'
73     fmt = logging.Formatter(fmt = common_format, datefmt = date_format)
```

```
74      || При автономной работе модуля устанавливаем уровень подробности вы-
        || вода сообщений и обработчика и логгера в DEBUG
76      log.setLevel(logging.DEBUG)
77      hand = logging.StreamHandler()
78      hand.setFormatter(fmt)
79      hand.setLevel(logging.DEBUG)
80      log.addHandler(hand)
```

6.2.2 Инициализация констант, загрузка модулей

Именованные поля для конструирования типа запроса. На основе CyAPI.h, Cypress USB библиотеки под виндоус.

```
85 TGT_DEVICE, TGT_INTFC, TGT_ENDPT, TGT_OTHER = 0, 1, 2, 3
86 REQ_STAD, REQ_CLASS, REQ_VENDOR = 0, 32, 64
87 DIR_TO_DEVICE, DIR_FROM_DEVICE = 0, 128
88      || Камак константы. Бит запрета, очистки, инициализации крейта и про-
        || граммного сброса контроллера.
90 CAM_I, CAM_C, CAM_Z, CAM_Ax4 = 0x04, 0x02, 0x01, 0x40
91      || Ожидание, при работе с последовательностью команд
92 NoWait, WaitForQ, WaitForL = 0, 1, 128

94 import sys, os, time, usb
```

6.2.3 Поиск устройства

Осуществляется поиск устройства. Программа сообщает о найденном адаптере без firmware или с firmware. Если адаптер не найден или найден адаптер без прошивки, программа завершает работу.

```
101 cypress_non_init = usb.core.find(idVendor = 0x04B4, idProduct = 0x8613)
102 if cypress_non_init is not None:
103     log.error(u"Найден неинициализированный адаптер usb-cc217.10")
104     log.error(u"Добавьте в систему автоматическую инициализацию.")
105     sys.exit()

107 cypress = usb.core.find(idVendor = 0x0547, idProduct = 0x1002)
108 if cypress is None:
109     log.error(u"Не найден адаптер usb-cc217.10")
110     sys.exit()
111 else:
112     log.info(u"Найден инициализированный адаптер usb-cc217.10")
113     try:
114         | Документация к pyusb (API 1.0) гласит, что при наличии одной
          | конфигурации указывать её не обязательно, но функцию установки
          | конфигурации надо вызывать. В то же время, при наличии одного
          | интерфейса можно даже не вызывать функцию, которая его уста-
          | навливает. [1] Однако при этом повторные загрузки модуля не сооб-
          | щают об ошибке Resource Busy, то есть возможна ситуация, когда
          | два экземпляра программы управления будут конкурировать за мо-
          | дуль. Поэтому сохраняем установку интерфейса и альтернативного
          | интерфейса.
123         cypress.set_configuration(1)
124         cypress.set_interface_altsetting(interface = 0, alternate_setting = 0)
125     except usb.core.USBError, msg:
126         log.error(u"Произошло исключение! причина: %s" % msg)
127         sys.exit()
```

6.3 VendRquToDevice, VendRquFromDevice, VendRqu

Подробности по аргументам в описании функции, осуществляющей основную работу: ctrl_transfer()

```
133 def VendRquToDevice(bRequest, wValue = 0, wIndex = 0, data =
        bytearray((0,)), timeout = 100):
```

```
134     Пользовательский запрос с передачей данных к устройству.

        Аргументы: data: данные для записи (array)
        Возвращает: количество записанных байт

139     rqType = TGT_DEVICE | REQ_VENDOR | DIR_TO_DEVICE   # 0x40
140     return cypress.ctrl_transfer (rqType, bRequest, wValue, wIndex, data, timeout)

143 def VendRquFromDevice (bRequest, wValue = 0, wIndex = 0, wLength =
        0, timeout = 100):
144     Пользовательский запрос с передачей данных от устройства.

        Аргументы:
            wLength: кол-во байт для чтения (целое)
        Возвращает: прочитанные байты

150     rqType = TGT_DEVICE | REQ_VENDOR | DIR_FROM_DEVICE   # 0xC0
151     return cypress.ctrl_transfer (
            rqType, bRequest, wValue, wIndex, wLength, timeout)

154 VendRqu = VendRquToDevice   # Более короткая форма для записи
        пользовательского запроса к устройству.
```

6.4 encode(D, A, F, N, WaitFor), подготовка для загрузки в адаптер

Аргументы: D целое в диапазоне от 0 до $2^{24} - 1 = 16777215$, A субадрес, F функция, N номер станции. WaitFor может быть одной из констант: NoWait $= 0$, сразу приступить к выполнению следующей команды; WaitForQ $= 1$ команда повторяется, пока не поступит подтверждение по Q, WaitForL $= 128$ ожидание LAM.

Функция раскладывает D на три байта, Возвращает подготовленный для загрузки в адаптер набор: (Hi, Md, Lo, $A, F, N, 0$, WaitFor), где Hi старший байт данных, Md средний байт данных, Lo младший байт данных, 7-й байт всегда ноль (незначащий), для остальных полей берутся аргументы функции без именений.

Примечание: Если D>16777215, то разложение всё равно происходит и выводится предупреждение.

```
176 def encode(D, A, F, N, WaitFor):
177     Подготовка записей для загрузки в адаптер.
178     if D > 1 << 24 - 1:
179         log.warning(u"Превышен максимально допустимый размер данных для записи")
180     (DD, Lo) = divmod(D, 256)
181     (DD, Md) = divmod(DD, 256)
182     (DD, Hi) = divmod(DD, 256)
183     return (Hi, Md, Lo, A, F, N, 0, WaitFor)
```

6.5 decode(sequence, i=-1), расшифровка данных адаптера

Адаптер по каждой по каждой выполненной команде даёт информацию: $(A, F, N, \mathsf{LXQ}, \mathsf{Hi}, \mathsf{Md}, \mathsf{Lo}, S)$ (содержит 8 элементов, каждый размером 1 байт). Здесь A субадрес, F функция, N номер станции, LXQ регистр статуса, который включает битовые поля:

7 L=0 если есть хоть один LAM-сигнал (1 бит)

2..6 LAM номер станции при наличии LAM-сигнала (5 бит) (LAM="look at me", требование каких-то действий)

1 X=0 если команда не прошла по каким-либо причинам (1 бит)

0 Q=1 станция выполнила команду (1 бит)

Далее, Hi старший байт прочитанных данных, Md средний байт, Lo младший байт. S была ли команда завершена или прекращена по таймауту. Если младший бит S равен 1, то было успешное завершение.

Процедура decode() принимает последовательность, склеенную из произвольного количества вышеописанных записей и каждую преобразует в запись вида

$(A, F, N, L, \mathsf{LAM}, X, Q, S, D)$, (содержит 9 элементов). Часть переменных повторяет входные аргументы, смысл переменных L, LAM, X, Q должен быть понятен из описания битовых полей регистра LXQ, данные собираются из байтов в число по формуле $D = (\mathsf{Hi} * 256 + \mathsf{Md}) * 256 + \mathsf{Lo}$

Необязательный парамер i служит для возможности добавления номера итерации в лог-файл.

```
215 def decode(sequence, i = -1):
216     | Преобразование результатов выполнения команд
217     log.debug(u"чтение␣%3i,␣A,␣␣F,␣␣N,␣␣L,LAM,␣␣X,␣␣Q,␣␣S,␣␣данные" %
                i)
218     chunks = []
219     for i in range(len(sequence)/8):
220         chunks.append(tuple(sequence[i * 8: (i + 1) * 8]))
221     result = []
222     for (A, F, N, LXQ, Hi, Md, Lo, S) in chunks:
223         L, LAM, X, Q = LXQ >> 7 & 1, LXQ >> 2 & int('11111', 2), LXQ >>
                    1 & 1, LXQ & 1
224         res_tuple = (A, F, N, L, LAM, X, Q, S & 1, (Hi * 256 + Md) * 256 + Lo)
225         log.debug(u"␣␣␣␣␣␣␣␣␣␣␣%2i,␣%2i,␣%2i,␣%2i,␣%2i,␣%2i,␣%2i,␣%2i,␣%i"
                    % res_tuple)
228         result+ = [res_tuple,]
229     return result
```

6.6 sequencer, класс для работы с последовательностями команд

В покомандном режиме каждый цикл обращения к КАМАК контроллеру и получения ответа от него занимает до 500мкс, это ограничение накладывает стандарт USB. В то же время адаптер способен работать с гораздо большей скоростью. Секвенсор даёт возможность загрузить последовательность команд и прочитать последовательность результатов их выполнения.

```
238 class sequencer(object):
239     def __init__(self, sequence):
240         self.InEndPt, self.OutEndPt = 0x86, 0x02
```

```
241         self.sequence = sequence
242         self.write (sequence)
```

6.6.1 write(self, sequence), Загрузка секвенсора

Формат записей для секвенсора см. в функции encode() Для повторной загрузки последовательности нужна инициализация адаптера (отключение и подключение к шине USB). Без этого попытка повторной загрузки приводит к таймауту. Поэтому при выходе по таймауту выдается предупреждение, в котором рекомендуется перезагрузить адаптер.

Первое чтение результатов возвращает саму загруженную последовательность. Проверяем её на совпадение с той, которая была загружена.

```
254     def write (self, sequence):
255         """ Загрузка списка КАМАК-команд в адаптер.
            Аргументы:  (Hi, Md, Lo, A, F, N, 0, WaitFor) * N
            Возвращает: ---
259         VendRqu (0xB3)
260         if (len (sequence)/8) * 8 != len (sequence):
261             log.error (u"Число байт в последовательности не кратно 8!")
262             sys.exit ()
263         else:
264             log.debug (u"загружается %i команд" % (len (sequence)/8, ))
265         self.length = len (sequence)
266         try:
267             cypress.write (self.OutEndPt, sequence, 0, timeout = 1000)   #
                    загрузка
268         except usb.core.USBError, msg:
269             log.error (u"Произошло исключение! причина: %s" % msg)
270             if msg[0] == u"Operation timed out":
271                 log.error (u"Для повторной загрузки последовательности")
272                 log.error (u"отключите и вновь подключите адаптер к шине USB!")
273             sys.exit ()
274         VendRqu (0xB6)
275         cmp_sequence = cypress.read (self.InEndPt, self.length)
276         if bytearray (sequence) != bytearray (cmp_sequence):
```

```
277                 log.error (u"Загруженная и прочитанная последовательности не совпадают!")
278                 sys.exit ()
279         self.start ()
```

6.6.2 start(self), запуск последовательности

```
282     def start (self): VendRqu (0xD1)
```

6.6.3 stop(self), останов работы последовательности

```
285     def stop (self): VendRqu (0xD2)
```

6.6.4 read(self), чтение результатов

Возвращает последовательность результатов выполения команд. Формат записей см. в документации к функции decode():

```
291     def read (self):
292         """
            Аргументы: ---
            Возвращает (A, F, N, LXQ, Hi, Md, Lo, S) * N
            подробное описание в документации функции decode()
            """
297         return cypress.read (self.InEndPt, self.length)
```

6.6.5 Camac(N, A, F, D), выполнение одной команды

Подробное описание аргументов и результата в функциях encode() и decode()

```
303 def Camac (N, A, F, D = 0):
```

```
304     Функция выполняет одну команду KAMAK.
        Аргументы: N, A, F, D
        Результат: A, F, N, LXQ, Hi, Md, Lo, S
        Подробное описание в документации функций encode() и decode()
309     VendRquToDevice(0xB7, data = encode(D, A, F, N, 0))
310     return VendRquFromDevice(0xB9, wLength = 8)
```

6.7 Функции инициализации крейта

```
313 def CamSetZ():
314     Инициализация, установка бита Z
315     VendRqu(0xB2, data = bytearray((CAM_Z,)))   # 0x01

317 def CamSetC():
318     Очистка, установка бита C
319     VendRqu(0xB2, data = bytearray((CAM_C,)))   # 0x02

321 def CamSetI():
322     Запрет, установка бита I
323     VendRqu(0xB2, data = bytearray((CAM_I,)))   # 0x04

325 def CamReset():
326     Программный сброс контроллера, установка бита Ax4
327     VendRqu(0xB2, data = bytearray((CAM_Ax4,)))   # 0x40
```

6.7.1 Выполнение инициализации

```
330 CamReset()
331 CamSetZ()

334 if __name__ == "__main__":
```

335

6.8 Команды, выполняемые при автономном запуске

В случае, если файл запущен как программа, выполняем команды

338 *decode* (*Camac* (0, 0, 0, 0))

339 *decode* (*Camac* (22, 0, 0, 1))

341 sequence = (*encode* (0, 3, 25, 8, NoWait) + *encode* (0, 12, 1, 8, WaitForQ) +
encode (0, 4, 0, 8, WaitForQ))

346 s = *sequencer* (sequence)

347 **for** i **in** *range* (3): *decode* (s.*read* (), i)

349 log.*info* (u"Загружен␣модуль␣usbcc.py")

Список литературы

[1] Programming with pyusb 1.0. http://pyusb.sourceforge.net/docs/1.0/tutorial.html. 12

[2] Universal serial bus revision 2.0 specification. http://www.usb.org/developers/docs. 9, 10

[3] Дмитрий Чекунов. Программисту usb-устройств. Часть 1. Знакомство с usb. *Современная электроника*, 10:68–71, 2004. 9, 10

[4] Дмитрий Чекунов. Программисту usb-устройств. Часть 2. Стандартные требования usb. *Современная электроника*, 12:70–75, 2004. 9