opencbm 0.4.0 Users Guide

Michael Klein, *nip@c64.org* , Spiro Trikaliotis, *cbm4win@trikaliotis.net* , Wolfgang Moser *d81.de* 2006-04-28

This document describes the opencbm package, a Linux kernel module and Windows kernel mode driver, and a few user space support programs to control and use serial devices as used by most Commodore (CBM) 8-bit machines.

Contents

1 Overview

The popular Commodore 8-bit home-computers like the C-64 and the VIC-20 are using a custom serial bus to talk to attached devices (disk drive, printer). The opencbm kernel module provides an interface to this so-called IEC bus at the level of simple TALK and LISTEN commands, similar to the one provided by the Commodore kernel routines. Additionally, some higher and lower level bus control is available as well, allowing for full control of the bus. The serial devices are connected to the PC's parallel port via an XM1541 or XA1541 cable and, optionally, an XP1541 or XP1571 add-on cable. For cables, cf. 1.4 (cable).

1.1 Introduction to opencbm

This is version 0.4.0 of opencbm, a kernel device driver for the serial CBM bus (C64, VIC-20, etc.) for Linux and Windows. opencbm is a re-join of the two projects cbm4linux (latest standalone version: 0.3.2) and cbm4win (latest standalone version: 0.1.0a). It should be noted that both projects were highly related from the beginning, as cbm4win 0.1.0 was based on cbm4linux 0.3.2.

Opencbm should work with any devices that understand the "normal" talk and listen commands of the CBM IEC bus. It has been tested with several 1541, 1541-II, 1571 and 1581 drives, and a MPS-1200 pritnter. 1541 clones like the Oceanic OC-118 have also been reported to work.

The following cable types are supported:

- XM1541 and XA1541 (cbm4linux version >= 0.2.1, cbm4win version >= 0.1.0)
- XP1541 (cbm4linux version >= 0.2.0, cbm4win version >= 0.1.0)
- XP1571 (cbm4linux version >= 0.2.4, cbm4win version >= 0.1.0)
- Modified XE1541 (only on Linux, obsoleted by the XM1541, see 'LINUX/config.make')

More information on the different cable types can be found in 1.4 (cable).

This package is provided 'as is', no warranty of any kind will be taken for any damage or data loss caused by it or by any use of it.

**** WARNING ***

HOTPLUGGING can KILL your hardware.

Do not connect anything to the parallel port while the system or a drive is up.

Always SHUT DOWN, CONNECT, REBOOT.

Again, absolutely NO WARRANTY.

****

1.2 Supported operating systems

opencbm supports the following operating systems:

- Linux 2.4 and 2.6; 2.0 and 2.2 might still work, but have not been tested for ages. For Linux, i386 and AMD64 architectures are supported.
- Windows NT 4.0, 2000, XP and Server 2003. The i386 architecture is fully supported; additionally, there are experimental AMD64 and iA64 versions available.

1.3 Supported CBM hardware

Currently, opencbm supports the following CBM devices:

- VIC 1541 (all variants, including VIC 1540 and clones)
- VIC 1570
- VIC 1571
- VIC 1581 (not with d64copy, not with cbmformat or cbmforng)
- other CBM IEC drives, printers, and compatibles (only with cbmctrl)

1.4 Cables

A standard X(E)1541 cable won't work with _this_ driver. In fact, there will probably never be a multitasking OS which works with one of these, that's why we call it XM1541, M for Multi tasking. Anyway, if you have a XE1541, the necessary modification is simple:

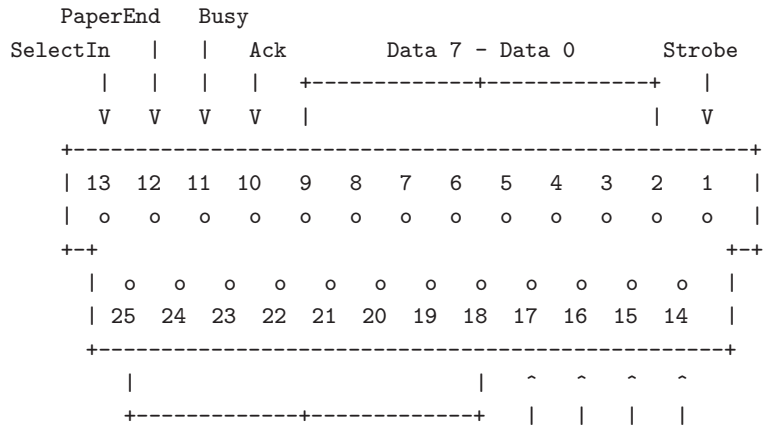
Exchange pins 5 & 6 on the Commodore DIN plug

The ACK line is the only line on a PC parallel port that can generate a hardware interrupt. This way, we get an interrupt when the device releases the DATA line to signal "ready to receive". Without an interrupt, you would have to poll for this signal about every 100us, which is inacceptable for any multitasking system.

Be sure to have your parallel port configured to use an IRQ, usually 7 or 5, but both are often also used by soundcards.

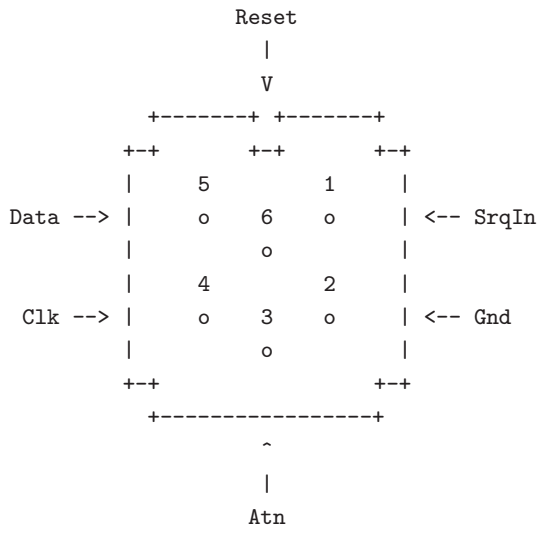
(ASCII art taken from the StarCommander README :))

The PC parallel plug (male DB-25 connector):



```
                    Ground          Select   |   |   AutoFeed
                                           Init   Error
```

The Commodore drive serial bus plug (male 6-pin DIN connector) looks like:



This is the XE1541 cable (won't work with this driver):

```
CBM drive serial port   PC parallel port

    2  Gnd ---------- 18-25  Ground
    3  Atn --------+---- 13  SelectIn
                   +->|-- 1  Strobe
    4  Clk --------+---- 12  PaperEnd
                   +->|- 14  AutoFeed
    5  Data -------+---- 11  Busy
                   +->|- 17  SelectIn
    6  Reset ------+---- 10  Ack
                   +->|- 16  Init
```

This is the XM1541 (pins 5 & 6 on the CBM end exchanged)

```
CBM drive serial port   PC parallel port

    2  Gnd ---------- 18-25  Ground
    3  Atn --------+---- 13  SelectIn
                   +->|-- 1  Strobe
    4  Clk --------+---- 12  PaperEnd
                   +->|- 14  AutoFeed
    6  Reset ------+---- 11  Busy
                   +->|- 17  SelectIn
    5  Data -------+---- 10  Ack
                   +->|- 16  Init
```

Besides the XM1541, a XA1541 cable is also supported. That cable consists of the same connections as the XM1541, but instead of using diodes, it uses transistors which drive the lines better. Because of this difference, the logic for outputs is reversed between the XA1541 and the XM1541.

Additionally to the cable types above, opencbm also supports XP1541 and XP1571 parallel cables, which have to be used *in conjunction* with the XM1541 or XA1541 cable.

For more information about the different supported cables (XM1541, XA1541, XP1541, XP1571) can be obtained on the Star Commander homepage (*http://sta.c64.org/xcables.html*)

2 News/Changelog

opencbm v0.4.0:

- General:
  - Reorganized structure so cbm4win and cbm4linux compile from the same sources
  - Fixed many minor and major errors
  - Added mnib36 (*http://rittwage.com/c64pp/dp.php?pg=mnib*) support
  - *cbmforng*: New tool, cf. 5.4 (cbmforng)
  - *rpm1541*: New tool, cf. 5.7 (rpm1541)
- General, Windows specific:
  - Use a free build instead of a checked build. This significantly reduces the memory footprint.
  - compiles for AMD64, iA64, i386 (Windows only)
  - VDD to allow DOS programs to access cbm4win
  - new unit file for Delphi, to allow to access cbm4win from Delphi
  - New project opencbmvice for debugging with the help of VICE (*http://www.viceteam.org/*). For this, a special version of VICE is needed.
- Linux driver:
  - Fixed kernel source directory (Dirk Jagdmann)
  - Fixed installation with GNU coreutils head (Dirk Jagdmann)
  - Added correct module installation dir for Linux 2.6 (Dirk Jagdmann)
  - Added descriptions for module parameters (*modinfo cbm*) (Dirk Jagdmann)
  - Added "smart reset" for cbm4linux: Delay the reset until all drives are ready.
- Windows driver:
  - Only access the bus if the parallel port was successfully acquired.
  - Added ECP and EPP support into NT4 driver (allowing XP1541 cable to be used there)
  - On reset, do not wait a fixed timeout anymore, just wait until all drives are ready again
- instcbm:
  - *–lock*, *–cabletype*: New options
  - *–automatic* is default now, new option *–on-demand* for old behaviour
  - Added *-V* (*–version*) command-line option
  - Reworked start of driver. It was unloaded and loaded before, which does not make sense
- cbmctrl:
  - *cbmctrl popen*, *cbmctrl pcommand* to do ASCII -> PETSCII conversions
  - *cbmctrl status*, *cbmctrl dir*: Output the status on stdout, not stderr
  - *cbmctrl lock*, *cbmctrl unlock*: New commands
  - *cbmctrl read*, *cbmctrl write*: New commands
  - Added *–version* and *–help* command-line arguments.

  - *cbmctrl change drive*: New function (heavily based on Joe Forster/STA's "TDCHANGE" from SC, used with permission)
  - *cbmctrl detect* outputs whether we have a parallel cable
- cbmcopy:
  - Fixed some timing problems which resulted in hanging in rare cases;
  - Cosmetical fix: The device status is written on a separate line on exit.
  - Fixed some races between PC and drive code in the transfer functions serial1, serial2, parallel
  - New option *–transfer=auto*, which is default and finds out the best transfer method for the current setup.
  - Do not use $14 in the floppy drive as temporary variable, but $86. This fixes a problem with Rex-DOS.
  - Do not trash the file on the PC side if aborted with Ctrl+C.
- d64copy:
  - Fixed some timing problems which resulted in hanging in rare cases;
  - *–warp* is default now; New option *–no-warp* for disabling it.
  - did not recognize .d71 files as valid images; fixed that.
  - Fixed some races between PC and drive code in the transfer functions *serial1*, *serial2*, *parallel*
  - New option *–transfer=auto*, which is default and finds out the best transfer method for the current setup.
  - Do not use $14 in the floppy drive as temporary variable, but $86. This fixes a problem with Rex-DOS.
  - Do not trash the file on the PC side if aborted with Ctrl+C.
- API:
  - `cbm_detect_xp1541()`: New function
  - `cbm_iec_setrelease()`: New function
  - `cbm_iec_set()`, `cbm_iec_release()`: Extended API to allow setting/resetting more than one line at the same time
- Build process (Windows):
  - reworked build process (*DDKBUILD_START.BAT*)
  - *DDKBUILD_LOCAL.BAT* contains settings for the CC65 build process, now.
  - *ddkbuild_local.bat.sample* added as sample for a DDKBUILD_LOCAL.BAT file
  - *postbuild_local.bat.sample* added as sample for a POSTBUILD_LOCAL.BAT file
- Build process (Linux):
  - Moved makefiles into LINUX directory; thus, use *make -f LINUX/Makefile* to compile now.

cbm4linux 0.3.3 (NEVER RELEASED!)

- documentation in *–help* for *d64copy* and *cbmcopy* fixed: now, it is clearly stated that a XP cable must be used in combination with a serial cable, not as only one. (Spiro Trikaliotis)
- fixed crash with unkown long options in d64copy, cbmformat and maybe cbmcopy (Spiro Trikaliotis)
- *cbmctrl upload* accepts - as filename now (read from stdin)
- *cbmctrl download* takes optionally a file name argument (Spiro Trikaliotis)
- *libd64copy* failed to recognize .d71 images as valid images. Because of this, you could not write a .d71 image back to a real floppy drive

- *d64copy*: If you copy a disc to an image which already exists, the error information was not removed from the file if necessary. This is fixed now. (Spiro Trikaliotis)
- *libd64copy*: Fixed a crash on exit of d64copy if a .d64 file grows.
- `parport_enumerate()`-fix for kernels>=2.6.4
- new ioctl `CBMCTRL_CLEAR_EOI` and API `function cbm_clear_eoi()` (Robert Norris)
- minor (still compatible) API changes (Spiro Trikaliotis)
- *cbmformat*: make sure disk name is 0-terminated (Spiro Trikaliotis)

3 Installation

Depending on the system you are running opencbm on, there are different ways to install opencbm. Use the appropriate category for you:

3.1 Installing opencbm on Linux (cbm4linux)

The kernel module (cbm.o) does not require any kernel patches and should compile right out of the box, at least with kernel 2.2.x and 2.4.x, but 2.0.x might still work as well.

If you intend to modify the drive routines for 'd64copy' and 'cbmformat' you also need a crossassembler. 'LINUX/config.make' comes with rules for A.Fachat's 'xa' (available from *http://www.floodgap.com/retrotech/xa/* or *http://www.lb.shuttle.de/puffin/cbm4linux/* ; Note: xa has not been tested lately, and might not work anymore) and Ullrich von Bassewitz' 'cl65' (comes with cc65, *http://www.cc65.org/*). Starting with version cbm4linux 0.2.3, opencbm includes precompiled 6502 binaries, so as long as you don't touch the .a65 files, there's no need for a crossassembler.

This package comes with a .spec file for those who want to build binary .RPMs. See the RPM documentation (outside of this paper) for details about the build process. Additionally, all files needed to built Debian .DEB packages are included. If you upgrade from a previous (non-RPM and non-DEB) version and want to install a packetized binary version (RPM or DEB), don't forget to remove the old files hanging aroung (just do "*make uninstall*", preferably in the *old* source directory. For a >= 0.4.0 version of opencbm, change the line to "*make -f LINUX/Makefile uninstall*".).

3.1.1 Compile-time configuration

The compile-time configuration is located in 'LINUX/config.make'. Check the KERN_FLAGS line if you're running kernel 2.0.x or if you don't want to use the Linux parport subsystem for some reason. Same goes for SMP machines.

3.1.2 Compilation

Type

- *make -f LINUX/Makefile* (no root privileges required)

  to build the kernel module, libraries and utility programs (no root privileges required),

- *make -f LINUX/Makefile dev* (as root)

  to create the character device "/dev/cbm" with major 10 and minor 177 (this number is registered, so it shouldn't collide with anything else :)). Finally

- *make -f LINUX/Makefile install* (as root)

  will install all necessary stuff to /usr/local/... (can be changed in 'LINUX/config.make')

3.1.3 Loading the module

If you're using the parport subsystem (which is default), you should now be able to load the driver module by issuing (as root)

- */sbin/depmod*
- */sbin/modprobe parport* (unless compiled into the kernel)
- */sbin/insmod cbm lp=*your_lp (usually 0, which is default)

or, when built with -DDIRECT_PORT_ACCESS:

- */sbin/insmod cbm port=your_ioport irq=your_irq* (default is 0x378 for port, 7 for irq)

Check `/var/log/messages` if the correct cable type was recognized (XA1541/XM1541).

3.1.4 Troubleshooting

Finding the cause of a failure condition can be hard. Anyway, the following tips might help you:

- Check `/var/log/messages`; it might give you some hints.
- If you are using the parport subsystem (no -DDIRECT_PORT_ACCESS):
  - the port might be occupied by another device (e.g. 'lp.o') cbm.o does NOT support port sharing (wouldn't work anyway). Enter *cat /proc/parport/port/devices* to find out.
  - parport_pc might not use an IRQ. */etc/modules.conf* should contain something like:

    ```
    alias parport_lowlevel parport_pc
    options parport_pc io=0x378 irq=7
    ```

    Check the interrupts with *cat /proc/interrupts*.
- Using direct port access (with -DDIRECT_PORT_ACCESS):
  - The port/IRQ might occupied by another driver (e.g. parport.o) Enter *cat /proc/interrupts* and *cat /proc/ioports* to find out.

3.1.5 Device access

As a first test, try something simple like

- *cbmctrl command 8 I0:* (assuming drive 8)
- *cbmctrl status 8*

(no root privileges required)

Failure can be caused by:

- Possibly, the shared library in `/usr/local/lib/` cannot be found; in this case, add `/usr/local/lib/` to `/etc/ld.so.conf` and execute *ldconfig* (as root)
- You might not have the necessary rights to the `/dev/cbm` device; try *chmod 777 /dev/cbm*
- incorrect module parameters
- wrong BIOS settings (esp. IRQ)
- broken cable

3.1.6 Runtime configuration

Most probably, you will want to add this to `/etc/modules.conf` to have the driver loaded on demand: (the file is called `/etc/conf.modules` on some older SuSE systems)

```
alias char-major-10-177 cbm
options cbm [options]
```

With [options] being one or more of:

- *lp=*lp** (parport only, as used in */sbin/insmod* above)
- *irq=*irq** (direct port only, as used in */sbin/insmod* above)
- *port=*port** (direct port only, as used in */sbin/insmod* above)
- *cable=*n** force cable type:
  - -1 for autodetection (default)
  - 0 for XM1541 (non-inverting)
  - 1 for XA1541 (inverting)
- *reset=*n** initializing behaviour:
  - -1 smart IEC reset (direct port default); only change the status of the reset line if it was set on start
  - 0 no IEC reset on driver start
  - 1 force IEC reset (parport default); always reset the device on driver start

Congratulation, you have successfully set up your opencbm installation!

3.2 Installing opencbm on Windows (cbm4win)

WARNING! If you have already installed a previous version of CBM4WIN on your machine, you have to uninstall it before installing a new version. For this, go to the directory where the old version is located, and enter instcbm –remove.

First of all, Windows must know about the driver. For this, we must install it with the instcbm tool. This is done as follows:

- Make sure you have a supported operating system up and running.
- You need administrator privileges on the Windows machine to perform the following actions.

- At first, you have to make sure you have the needed hardware ready. Do the following:
  - Get your supported drive 1.3 ().
  - Moving cables with equipment turned on can damage either your PC, and/or the drive, so, be carefull!
  - Thus, switch off your PC and your VIC 15xx drive!
  - Connect your XA1541 or XM1541 cable to your PC. If you have a parallel port cable (XP1541), connect that one, too.
  - Connect your VIC 15xx floppy drive to the cable
  - Switch on the PC.
- Just download the binary package, and unpack it into an arbitrary directory.
- Get a command-line (Start/Run, and type "*cmd.exe*"), change into the directory you unpackaged the drivers into (with "*cd*").
- Type "*cd exe*"
- Type "*instcbm*" and check the outputs. Its last line should look like *No problems found in current configuration*. In this case, you are done. In some rare cases, *instcbm* will suggest a reboot, which you should follow.
- You might want to have a look at the possible options for instcbm. They are available by typing "*instcbm –help*". Also, cf. 5.1 (instcbm).
- If you had to reboot in the previous step, do the following:
  - Go to a command-line, and change into the directory you unpackaged the drivers into again.
  - Type "*cd exe*"
  - Type "*instcbm –check*". There should not be any further suggestion for a reboot. If there is, do not proceed, but contact me instead.
- If you want to use another port than LPT1, you must tell this to the driver. I assume you want to use LPTX, with X being the correct value, then type: "*instcbm –lpt=X –update*"

4 Checking if the installation is complete

After you installed opencbm (cf. 3 (Installation), it is wise to check if the installation works as expected. For this, do the following:

- Switch on the floppy drive. Depending on the type of cable you are using (XA1541 or XM1541) and the parallel port of your PC, the drive might keep spinning endless now, because it is continuously resetted.
- Type "*cbmctrl reset*" and press enter. If it does not already, the red floppy drive LED should light up, and the drive should start spinning. After approximately one second (up to five seconds in the case of a 1581), the red LED should switch off again, and the drive stops spinning.
- Now, type "*cbmctrl status 8*" to get the status (error) code from the attached floppy drive. If everything works fine, your drive should answer with its identification string. For a 1541, this is something like *73,cbm dos v2.6 1541,00,00*, while for a 1571, this line looks like *73,cbm dos v3.0 1571,00,00*. There might also be some variant of this line, depending on the firmware version of your drive.

- Type "*cbmctrl status 8*" to get the status (error) code from the floppy drive again. As the power on message has been read, your drive should answer with a *00, ok,00,00* string.
- Type "*cbmctrl detect*". This command tries to detect the types of drive which are connected on the cable. You should see the drive which you posess.
- Now, we want to check if we can send anything to the floppy drive. Remove any diskette from the drive and press "*cbmctrl open 8 15 I0*". (Make sure the "I" is an upper-case "I". A lower-case "I" will not work!) This command tries to initialize the disk. Anyway, since there isn't a disk in the drive, an error occurs. You should hear the floppy spinning, and in case of a 1541, the R/W-head should start bumping. After some seconds, the red LED starts starts flashing, indicating that an error occurred.
- Now, try again "*cbmctrl status 8*" to get the status (error) code from the floppy drive. As an error occurred before, an error string should be displayed. For my setup, it is the "*21,read error,18,00*" string. Furthermore, the red LED should stop flashing.

If you have come so far, you are sure that you send commands to the floppy, and receive answers from it. This is very good so far. Furthermore, don't panic: you do not have to enter these commands over and over again, these are only tests to make sure that anything is correctly installed.

Now, let's proceed. If you have a D64 file or a floppy disc ready, you can try transferring it over the cable. Do not use all of the following commands, but only the ones you want to perform.

- If you want to transfer an existing floppy from the drive to the PC, use the following command: "*d64copy 8 A.D64*", while replacing A.D64 by the name you want to give to the file.
- **WARNING THE FOLLOWING COMMAND OVERWRITES ANYTHING THAT WAS ON THE FLOPPY BEFORE, so make sure you do not need that floppy anymore.**

  If you have a D64 or D71 on your PC, and you want to write it to a new, already formatted disc, enter "*d64copy A.D64 8*" if the file is called A.D64.
- **WARNING THE FOLLOWING COMMAND OVERWRITES ANYTHING THAT WAS ON THE FLOPPY BEFORE, so make sure you do not need that floppy anymore.**

  If you have a disc you want to format, you have two options: Either use the command "*cbmctrl command 8 N0:NAME,ID*", or use the cbmformat program, cf. 5.3 (cbmformat), or the cbmforng program, cf. 5.4 (cbmforng).

If you want to completely remove the cbm4win driver from your machine, you can do so by issuing a "*instcbm –remove*" command.

You can have a look at the available cbmctrl commands by issuing cbmctrl on your command line, or look at 5.2 (cbmctrl). For the other programs, you get help by issuing the "–help" option, or look at the appropriate section in 5 (utilities).

5 Utilities

As the kernel driver is quite useless for itself, the following utility programs are included with this package:

- *cbmctrl*

  command line utility for direct device access at talk/listen level.
- *cbmformat*

  fast 1541 disk formatter (for 1541, 1570 and 1571 drives).

- *cbmforng*

  fast 1541 disk formatter (for 1541, 1570 and 1571 drives).

- *d64copy*

  copies .d64 images to 1541 compatible drives and vice versa. Type 'd64copy -h' to get a list of valid options. Use the device number to address the disk drive, e.g. 'd64copy -t serial2 8 img.d64'.

  This version contains the StarCommander Turbo and Warp routines and custom transfer routines as well as parallel cable (XP1541) support. These are the benchmarks for Michael Klein's old Pentium-200/MMX system (seconds)

```
mode               write     read

parallel turbo      54.02    53.30
parallel warp       32.47    29.56
serial1 turbo      168.59   168.58
serial1 warp       183.57   158.87
serial2 turbo       95.02    95.26
serial2 warp        88.29    80.57
```

- *cbmcopy*

  fast 1541/1570/1571/1581 file copier.

- *rpm1541* demo

  determines the drive rotation speed of 1541, 1570 and 1571 drives.

- *flash* demo

  flashes the LED of 1541, 1570 and 1571 drives.

- *morse* demo

  morses arbitrary texts with the help of the LED of 1541, 1570 and 1571 drives.

5.1 instcbm (Windows only)

instcbm is used on Windows to install the opencbm driver.

5.1.1 instcbm invocation

Synopsis: `instcbm` *`[options]`*

-h, –help

display help and exit.

-V, –version

display version information about cbm4win.

-r, –remove

remove (uninstall) the driver.

-e, –enumpport

re-enumerate the parallel port driver.

-u, –update

update parameters if driver is already installed.

-l, –lpt=*no*

set default LPT port to number **no** . For example, for LPT2, use *–lpt=2*.

If not specified, or *–lpt=0* is specified, use the first parallel port.

-t, –cabletype=*TYPE*

set cabletype to **TYPE**, which can be *auto*, *xa1541* or *xm1541*.

If not specified, *–cabletype=auto* is assumed.

-L, –lock=*WHAT*

automatically lock the driver. **WHAT** can be *yes* (automatically lock) or *no* (do not automatically lock).

If not specified, *–lock=yes* is assumed.

-n, –nocopy

do not copy the driver files into the system directory. This is not recommended.

-c, –check

only check if the installation is ok. Do not install or uninstall anything.

-F, –forcent4

force the NT4 driver on a Win 2000, XP, or newer systems (NOT RECOMMENDED!). This option is only available on i386 architectures; AMD64 and iA64 do not support it.

-A, –automatic

(default) automatically start the driver on system boot.a

The driver can be used from a normal user, no need for administrator rights. The opposite of *–on-demand*.

-O, –on-demand

start the driver only on demand.

The opposite of *–automatic*.

5.1.2 instcbm Examples

Install the driver and the DLL on the machine. The driver and the DLL are copied into the Windows system directory, so opencbm can be used from every program:

```
instcbm
```

Install the driver, like above. Additionally, specify that you are using an XM1541 cable:

```
instcbm --cabletype=xm1541
```

Check if the installation was set up successfully:

```
instcbm --check
```

Remove the driver from the system. You will not be able to use opencbm after this command, unless you re-install it. If files were copied into the Windows system directory, they will be removed:

```
instcbm --remove
```

After opencbm has been installed (with *instcbm>*), change the parallel port to be used to 2:

```
instcbm --lpt=2 --update
```

Install opencbm, directly specifying LPT3 as the parallel port to use:

```
instcbm --lpt=3
```

Install the DLL and the driver on the machine. Do not copy the files to the Windows system directory, but leave them "where they are". If you use this option, the directory where your files resides must be accessible for the system while booting. For example, network drives, USB drives or FireWire drives are not allowed.

```
instcbm --nocopy
```

5.2 cbmctrl

cbmctrl is used to send commands to external devices. It can control all kinds of serial CBM devices like floppy drives and printers. So far, it has been successfully tested with the disk drives 1541(-II), 1571 and a MPS-1200 printer.

5.2.1 Command structure

The overall format of all *cbmctrl* actions is:

Synopsis: `cbmctrl` *`[global_options]`* `ACTION` *`[action_args]`*

global_options

Some options that are related to *cbmctrl* in general of which affect the oervall behaviour of all actions

action

One of a bunch of different subcommands that direct *cbmctrl* what to do

action_args

Arguments that are required for the subcommand *action* to work

Global options *cbmctrl* understands the following global options

-h *[ACTION]*, –help *[ACTION]*

Outputs the help screen with a short listing of all available actions. If the optional *ACTION* name is given also, you retrieve more information on a special action together with its arguments and parameters

-V, –version

Output version information as well as the built date and time

Actions overview *cbmctrl* understands the following subcommand actions

reset

Reset all drives on the IEC bus

detect

Detect all drives on the IEC bus

lock

Lock the parallel port for opencbm (cbm4linux/cbm4win) use

unlock

Unlock the parallel port from exclusive usage

listen

Perform a listen on the IEC bus

talk

Perform a talk on the IEC bus

unlisten

Perform an unlisten on the IEC bus

untalk

Perform an untalk on the IEC bus

open

Perform an open on the IEC bus

close

Perform a close on the IEC bus

popen

Same as open, but with ASCII to PETSCII conversion

read

Get a stream of raw data from an IEC bus device

write

Put a stream of raw data to an IEC bus device

status

Give the status of a specified drive

command

Issue a command to a specified drive

pcommand

Same as command, with ASCII to PETSCII conversion

dir

Output the directory of a disk in a specified drive

download

Download memory contents from a floppy drive

upload

Upload memory contents to a floppy drive

change

Wait for a disk to be changed in a specified drive

Common action arguments Many of the *cbmctrl* subcommands understand the following common arguments:

[**DEVICE**]

Advice *cbmctrl* to direct its communication to the IEC bus device with the number *[DEVICE]*. IEC bus device numbers can be denoted in the range from 0 to 30, although no Commodore device is known to use device numbers 0 to 3. Most commonly used are the numbers 4 (printer) and 8 to 11 (disk drives). Device number 31 is used to denote the UNTALK respectively the UNLISTEN command code on the IEC bus instead of the TALK respectively LISTEN command code, therefore device address 31 cannot be used in general.

[**SECADR**]

With several *cbmctrl* actions the secondary address parameter *[SECADR]* denotes a dedicated logical communication channel for the specifed *[DEVICE]* . IEC bus channel numbers can be denoted in the range from 0 to 15. Take note that for floppy disk drive devices some secondary addresses are interpreted in a special way. Secondary address 0 is used, when a program is loaded, address 1, when a program is saved. Address number 15 represents the command channel of the disk drive, so e®ectively, for bulk data transfers to and from disk drives, only the logical channel numbers 2 to 14 can be used.

5.2.2 Actions

cbmctrl understands the following actions:

reset

This action performs a hardware reset of all devices attached to the IEC bus. Control is returned after it is made sure that all devices are ready.

detect

This action tries to detect all devices attached to the IEC bus. For this, this subcommand accesses all possible devices and tries to read some bytes from its memory. If a devices is detected, its name is output. Additionally, this routine determines if the device is connected via a parallel cable (XP1541 companion cable, may be true for disk drives only).

lock

This command locks the parallel port for the use by opencbm, so that sequences of e.g. *talk/read/untalk or listenwrite/unlisten* are not broken by concurrent processes wanting to access the parallel port.

You should issue *cbmctrl lock* before doing any access to opencbm tools, and *cbmctrl unlock* after you are done.

unlock

This command unlocks the parallel port after the use by opencbm.

You should issue *cbmctrl lock* before doing any access to opencbm tools, and *cbmctrl unlock* after you are done.

listen *device secadr*

Tell device *device* to listen on secondary address *secadr*. Until the next *unlisten* command, everything output with *cbmctrl write* will be received by this device.

This command corresponds to the following 6502 assembly code on a C64:

```
lda #device
jsr $ffb1
lda #secadr
ora #$60
jsr $ff93
```

talk *device secadr*

Tell device *device* to talk on secondary address *secadr*. Until the next *untalk* command, data from this device can be received device by using the command *cbmctrl read*.

This command corresponds to the following 6502 assembly code on a C64:

```
lda #device
jsr $ffb4
lda #secadr
ora #$60
jsr $ff96
```

unlisten

Ends communication with listening devices after a *listen* command. This corresponds to the C64 kernel routine $ffae.

untalk

Ends communication with talking devices after a *talk* command. This corresponds to the C64 kernel routine $ffab.

open *device secadr filename*

Open file *filename* on device *device*. After opening, data can be read/written by sending a *talk* resp. *listen* command with the secondary address *secadr*.

If *secadr* is greater than 1, the file type and access mode must also be specified by appending `,type,mode` to *filename*. Valid types are D, P, S, U, R (DEL, PRG, SEQ, USR, REL), valid modes are R for reading and W for writing.

Note: You cannot do an open without a filename. Although a CBM machine (i.e., a C64) allows this, this is an internal operation for the Computer only. It does not have any effect on the IEC bus.

cbmctrl open does not change any character encoding, that is, it does not convert between ASCII (used by the PC) and PETSCII (used by the CBM device). If this is needed, use *cbmctrl popen* instead.

popen *device secadr filename*

Like *cbmctrl open*, but converts the filename from ASCII to PetSCII before sending it to the floppy.

close *device secadr*

Close the file associated with secondary address *secadr* on device *device*.

read *[file]*

This command reads raw data from the IEC bus and outputs it into the given file, or to stdout if no file is given (or if it is a simple dash, "-").

write *[file]*

This command writes raw data to the IEC bus; the data is taken from the given file, or from stdin if no filename is given (or if it is a simple dash, "-").

status *device*

Copies input from device *device*, secondary address 15 (command/status channel), to the standard output stream. Note that all upper case characters are changed to lower case. Carriage return (0x0d) is also changed to the current operating system's line ending convention (0x0a on Unix oriented systems, 0x0d 0x0a on Windows oriented systems or whatever else is appropriate for your operating system).

Assuming the device number is 8, this command is similar to (in this case, no character conversions would be made)

```
cbmctrl lock
cbmctrl talk 8 15
cbmctrl read
cbmctrl untalk
cbmctrl unlock
```

command *device cmdstr*

Sends *cmdstr* to device *device*, secondary address 15 (command/status channel). Since there is no PetSCII->ASCII conversion, commands must be sent in *upper case* (kind of poor man's PetSCII conversion). This is because charset conversion would break the `M-W` and `M-E` commands.

Note: If you need PetSCII->ASCII conversion, use *pcommand* instead.

Assuming the device number is 8, this command is identical to (Note: This does not work on Windows, because *echo* there does not know the *-n* option.)

```
cbmctrl lock
cbmctrl listen 8 15
echo -n cmdstr|cbmctrl write -
cbmctrl unlisten
cbmctrl unlock
```

pcommand *device cmdstr*

Like *command*, but converts the data from ASCII to PetSCII before sending it.

dir *device*

Read directory from disk in device *device*, print on standard out.

download *device address count [file]*

Read *count* bytes from drive memory, starting at *address* via one or more `M-R` commands. Memory contents are written to standard output if *file* is ommited or equivalent to `"-"`.

upload ***device address [file]***

Send *file* to drive memory, starting at *address* via one or more `M-W` commands. If *address* is -1, the first two bytes from *file* are considered as start address. Reads standard input if *file* is ommited or equivalent to `"-"`.

change ***device***

Wait for a disc to be changed in the specified device. It waits for the current disc to be removed, for a new disc to be inserted and for the drive door to be closed. It does not return until the disc is ready to be read or written.

5.2.3 cbmctrl Examples

Send file contents to printer 4:

```
cbmctrl lock
cbmctrl listen 4 0
cbmctrl write file
cbmctrl unlisten
cbmctrl unlock
```

Copy file to disk drive 8:

```
cbmctrl lock
cbmctrl open 8 2 FILENAME,P,W
cbmctrl listen 8 2
cbmctrl write file
cbmctrl unlisten
cbmctrl close 8 2
cbmctrl unlock
```

Copy file from disk drive 8:

```
cbmctrl lock
cbmctrl open 8 2 FILENAME,P,R
cbmctrl talk 8 2
cbmctrl read file
cbmctrl untalk
cbmctrl close 8 2
cbmctrl unlock
```

Dump 1541 ROM:

```
cbmctrl download 8 0xc000 0x4000 > 1541.rom
```

or

```
cbmctrl download 8 0xc000 0x4000 1541.rom
```

Write file buffer2.bin to drive 9, address 0x500:

```
cbmctrl upload 9 0x500 buffer2.bin
```

5.3 cbmformat

cbmformat is a fast low-level disk formatter for the 1541 and compatible devices (1570, 1571, third-party clones). A 1581 drive is not supported.

The drive routine was taken from the Star Commander ((C) Joe Forster/STA) and highly improved.

There is also another, very similar tool, 5.4 (cbmforng).

5.3.1 cbmformat invocation

Synopsis: `cbmformat [OPTION]... DRIVE# NAME,ID`

DRIVE# has to be the drive number of the disc drive, *NAME* is a name with up to 16 characters which will be the name of the disc after formatting, *ID* is the 2-letter disc ID.

Note: Unlike the *N0* command of the drive, the ID must be given (thus, no so-called "short format" is possible).

Here's a complete list of known options:

-h, –help

Display help and exit.

-V, –version

Display version information and exit.

-n, –no-bump

Do not bump drive head at the beginning. Don't use this on eventually misaligned drives.

-x, –extended

Format a 40 track disk, the BAM format is compatible to SpeedDOS.

-c, –clear

clear (demagnetize) this disc. This is highly recommended if the disc is used for the first time, or if it was previously formatted for another system (i.e., MS-DOS). Note that this option takes much time.

-v, –verify

verify each track after it is written. As this needs an extra round of the drive for each track, the formatting time is almost doubled.

cf. 5.3.2 (cbmformat Notes for 1571 drives)

-o, –original

Fill sectors with the original pattern (0x4b, 0x01, 0x01...) instead of zeroes. The original pattern is probably due to a bug in the drive ROM, apart from this, zeroing out unused sectors should give (slightly) better results for compressed disk images.

cf. 5.3.2 (cbmformat Notes for 1571 drives)

-s, –status

Display drive status after formatting. Normally, *cbmformat* exits after executing the drive code. With this option turned on, *cbmformat* waits until the drive has finished formatting and prints the drive status after initializing the BAM on standard out.

-p, –progress

Display a hash mark ('#') for each formatted track. Slows formatting down a bit.

5.3.2 cbmformat Notes for 1571 drives

We encountered problems with decent revision/mechanics combinations of the 1571 disk drives when using cbmformat. We highly recommend to use *–original* and *–verify* with 1571 drives. From our experience, with *–original*, the problem does not occur; with *–verify*, the drive tests each track after it was formatted and ensures that the failure condition did not occur.

We did not encounter these problems with either of 1541 (1541-II, 1541C), 1570 or 1571CR (the drive which is part of the C128DCR) drives, only with original 1571 drives.

In the current state, cbmformat is not able to format double-sided discs on a 1571 drive.

5.3.3 cbmformat Examples

Format standard disk (35 tracks) in drive 8:

```
cbmformat 8 GAMES,42
```

Format standard disk (35 tracks) in drive 9, use (buggy) 1541 sector pattern (for example, because this is a 1571 drive), show drive status when done:

```
cbmformat -os 9 1571disc,71
```

SpeedDOS disk (40 tracks), show progress indicator, all sectors zeroed out, no head banging:

```
cbmformat -npx 8 "40 TRACKS,OK"
```

5.4 cbmforng

cbmforng is a fast and reliable low-level disk formatter for the 1541 and compatible devices (1570, 1571, third-party clones). It was based on 5.3 (cbmformat) and is designed to become the designated successor to 5.3 (cbmformat), therefore its name: *CBM-Formatter, the Next Generation*.

cbmforng does not support a 1581 drive.

Because this is the first official release of *cbmforng* and because it was not used in the field by a wider user group, it still contains additional measurement routines and informational output after the formatting process was done. When *cbmforng* prooved its matureness and got back some features currently missing (progress bar), it will replace *cbmformat*.

To date *cbmforng* should be considered as the more reliable formatter of both; whenever you should encounter any difficulties with *cbmformat*, go for *cbmforng*. If you like additional informational messages like e.g. the RPM value each formatted track was measured, then *cbmforng* is the tool you want to use. Your feedback helps us to decide, if this additional output which was needed for developing may find its way into future releases.

5.4.1 cbmforng invocation

Synopsis: `cbmforng [OPTION]... DRIVE# NAME,ID`

DRIVE# has to be the drive number of the disc drive, *NAME* is a name with up to 16 characters which will be the name of the disc after formatting, *ID* is the 2-letter disc ID.

Note: Unlike the *N0* command of the drive, the ID must be given (thus, no so-called "short format" is possible).

Here's a complete list of known options:

-h, –help

Display help and exit.

-V, –version

Display version information and exit.

-n, –no-bump

Do not bump drive head at the beginning. Don't use this on eventually misaligned drives.

-r, –retries n

Set the maximum number of retries on errors. This accounts for all errors that may happen when formatting all the tracks of the whole disc.

-x, –extended

Format a 40 track disk, the BAM format is compatible to SpeedDOS.

-c, –clear

clear (demagnetize) this disc. This is highly recommended if the disc is used for the first time, or if it was previously formatted for another system (i.e., MS-DOS). Note that this option takes much time.

-v, –verify

verify each track after it is written. As this needs an extra round of the drive for each track, the formatting time is almost doubled.

cf. 5.4.2 (cbmforng Notes for 1571 drives)

-o, –original

Fill sectors with the original pattern (0x4b, 0x01, 0x01...) instead of zeroes. The original pattern is probably due to a bug in the drive ROM, apart from this, zeroing out unused sectors should give (slightly) better results for compressed disk images. In comparison to *cbmformat*, the pattern used with *cbmforng* is a little bit more original than the one from its predecessor. On track one the pattern consists of: 0x00, 0x01, 0x01, ... instead of the first byte beeing 0x4b. This perfectly reflects the original 1541 ROM format bug.

cf. 5.4.2 (cbmforng Notes for 1571 drives)

-s, –status

In addition to the informational output of internal values from the formatting process, the drive status is displayed.

5.4.2 cbmforng Notes for 1571 drives

We encountered rare failure conditions with decent revision/mechanics combinations of the 1571 disk drives when using cbmforng. We highly recommend to use *–original* and *–verify* with 1571 drives. From our experience, with *–original*, the problem does not occur. With *–verify*, the drive tests each track after it was formatted and ensures that the failure condition did not occur; otherwise the same track is formatted again, as often as the currently set retry value allows.

We did not encounter these problems with either of 1541 (1541-II, 1541C), 1570 or 1571CR (the drive which is part of the C128DCR) drives, only with original 1571 drives.

In the current state, cbmforng is not able to format double-sided discs on a 1571 drive.

5.4.3 cbmforng Examples

Format standard disk (35 tracks) in drive 8:

```
cbmforng 8 GAMES,42
```

Format standard disk (35 tracks) in drive 9, use (buggy) 1541 sector pattern (for example, because this is a 1571 drive), show drive status when done:

```
cbmforng -os 9 1571disc,71
```

SpeedDOS disk (40 tracks), verify formatted tracks, all sectors zeroed out, no head banging:

```
cbmforng -nvx 8 "40 TRACKS,OK"
```

5.5 d64copy

d64copy is a fast disk image transfer (both read and write) program for the 1541 and compatible devices (1570, 1571, third-party clones). A 1581 drive is *not* supported! Maximum transfer speed is achieved by custom drive- and transfer-routines based on the Star Commander ((C) Joe Forster/STA) routines.

5.5.1 d64copy invocation

Synopsis: `d64copy [OPTION]... SOURCE TARGET`

Either SOURCE or TARGET must be an external drive, valid names are 8, 9, 10 and 11. The other parameter specifies the file name of the .d64 image.

Here's a complete list of known options:

-h, –help

Display help and exit

-V, –version

Display version information and exit.

-q, –quiet

Quiet output, fewer messages (also suppresses warnings, should not be used)

-v, –verbose

Verbose output, more messages (can be repeated)

-n, –no-progress

Omit progress display

-s, –start-track=*start track*

Set start track (defaults to 1)

-e, –end-track=*end track*

Set end track (default is 35 for .d64 images, 70 for .d71 images). *d64copy* is able to access tracks 1-35 in `original` transfer mode and 1-42 with `serial1`, `serial2` and `parallel`. The 1571 supports tracks 1-70 in double sided (.d71) mode.

-t, –transfer=*transfer mode*

Set transfermode. Valid modes are:

- `auto` (default)
- `original` (slowest)
- `serial1`
- `serial2`
- `parallel` (fastest)

`original` and `serial1` should work in any case. `serial2` won't work with more than one device connected to the IEC bus, `parallel` requires an additional XP1541/XP1571 cable.

If `auto` is used, d64copy itself determines the best transfer mode usable with the current setup, and uses that one. Thus, you will seldom want to manually overdrive the *transfer mode* option.

-i, –interleave=*interleave*

Set interleave value. This is ignored when reading in warp mode. Default is 16 for transfer mode `original`, for turbo and warp write as follows:

	turbo (r/w)	warp (write only)
serial1	3	5
serial2	12	11
parallel	6	3

Lower values might slightly reduce transfer times, but if set a bit to low, transfer times will dramatically increase.

-w, –warp

Enable warp mode. This is default now; this option is only supported for backward-compatibility with opencbm (cbm4linux/cbm4win) versions before 0.4.0.

–no-warp

Disable warp mode. Warp mode is usually a good idea for transferring disk images unless you have a very slow CPU and/or bad disk material. Warp mode sends raw GCR data over the bus, which assures data integrity on the PC side and relieves the drive's CPU. Thus, it is unlikely you will want to use that option.

-b, –bam-only

BAM-only copy. Only blocks marked as allocated are copied. For extended tracks (36-40), SpeedDOS BAM format is assumed. Use with caution, at least one wide-spread directory editor tends to forget to allocate some directory blocks.

-B, –bam-save

Safe BAM-only copy. This is like the `-b` option but always copies the entire directory track (18, 18 and 53 in double-sided mode).

-d, –drive-type=`type`

Skip drive type detection. `0` or `1541` specifies 1541 mode (1 MHz, parallel cable at VIA $1800), `1` or `1571` forces 1571 mode (2 MHz, parallel cable at CIA $4000).

-2, –two-sided

Double-sided mode for copying .d71 images to/from a 1571 drive. Warp mode is not supported (yet).

-r, –retry-count=`count`

Number of retries.

-E, –error-mode=`mode`

Controls whether error is appended to the disk image (15x1->PC only). Allowed values for `mode` are (abbreviations allowed):

- `always`
- `on_error` (default)
- `never`

5.5.2 d64copy Examples

Read a D64 disc image from the floppy in drive 8 to the file image.d64, automatically selecting the fastest transfer method:

```
d64copy 8 image.d64
```

Copy the D64 disc image in image.d64 to the floppy in drive 9, automatically selecting the fastest transfer method:

```
d64copy image.d64 9
```

Copy a double-sided disc from a 1571 drive 9 to image.d71, using `serial1` transfer method and only reading the blocks which are marked as used in the BAM:

```
d64copy -2 -B --transfer=serial1 9 image.d64
```

5.6 cbmcopy

cbmcopy is a fast file transfer program for various disk drives, in particular the 1541, 1570, 1571 and 1581 devices. Maximum transfer speed is achieved by custom drive- and transfer-routines based on the Star Commander ((C) Joe Forster/STA) routines.

5.6.1 cbmcopy invocation

Synopsis: `cbmcopy [OPTION]... DEVICE# FILE...`

DEVICE# specifies the drive number for file copy. The remaining arguments specify the files to be sent to/read from the disk drive. This version supports Raw, PC64 (P00) and T64 files. They are recognized when sending files to the disk drive, files read from external devices are always stored as raw binary data.

Here's a complete list of known options:

-h, –help

Display help and exit

-V, –version

Display version information and exit.

-q, –quiet

Quiet output, fewer messages (also suppresses warnings, should not be used)

-v, –verbose

Verbose output, more messages (can be repeated)

-n, –no-progress

Omit progress display

-r, –read

Operate in read-mode, i.e. read data from an external device. Starting *cbmcopy* as *cbmread* has the same effect.

-w, –write

Operate in write-mode, i.e. send files to an external device. Starting *cbmcopy* as *cbmwrite* has the same effect.

-t, –transfer=*transfer mode*

Set transfermode. Valid modes are:

- `auto` (default)
- `serial1` (slowest)
- `serial2`
- `parallel` (fastest, not possible with a 1581)

`serial1` should work in any case. `serial2` won't work with more than one device connected to the IEC bus, `parallel` requires a XP1541/XP1571 cable in addition to the XM/XA1541. If `auto` is given, or this option is completely omitted, cbmcopy will automatically determine the fastest transfer method possible with the current setup. Thus, you will seldom want to manually overdrive the *transfer mode* option.

-d, –drive-type=`type`

Skip drive type detection. Valid types are `1541`, `1570`, `1571` and `1581`.

-o, –output=`name`

Specifies target name. ASCII/PetSCII conversion is performed when in write-mode.

-a, –address=`address`

Overrides the file's first two bytes with *address*.

-R, –raw

Skip file type detection. File data is sent as is. This option is only valid in write-mode.

-f, –file-type=`type`

Specifies/overrides file type. Supported types are `P`, `S`, `D`, `U`. Raw files default to `P`, whereas the T64 format contains meta data which includes the file type. For PC64 files, *cbmwrite* tries to guess the file type from the file extension. This option is only valid in write-mode.

5.6.2 cbmcopy Examples

Read a file called *cbmfile* from drive 8 and store its binary value into the file file.bin, automatically selecting the fastest transfer method:

```
cbmcopy -r 8 cbmfile -o file.bin
```

Write out the file file.p00 in P64 format to the disc in drive 9, using `serial1` transfer method:

```
cbmcopy -w 9 file.p00
```

5.7 rpm1541

rpm1541 is a demo program. It finds out the rotation speed (in rounds per minute, rpm) of the drive motor. *rpm1541* supports a 1541, 1570 or 1571 drive. A 1581 drive is *not* supported.

For Linux, *rpm1541* is not installed automatically. You have to compile it yourself (found in `demo/rpm1541/`) if you want to use it. For Windows, it is part of the binary distribution.

5.7.1 rpm1541 usage

Synopsis: `rpm1541` *`[device]`*

The optional parameter *device* is the device number of the drive which should be tested. If not specified, `rpm1541` utilizes drive 8.

5.7.2 rpm1541 Example

Find out the rotation speed of drive 11:

```
cbmctrl lock
rpm1541 11
cbmctrl unlock
```

5.8 flash

flash is a demo program. It flashes the drive LED. *flash* works with 1541, 1570 or 1571 drives. A 1581 drive is *not* supported.

For Linux, *flash* is not installed automatically. You have to compile it yourself (found in `demo/flash/`) if you want to use it. For Windows, it is part of the binary distribution.

5.8.1 flash usage

Synopsis: `flash` *`[device]`*

The optional parameter *device* is the device number of the drive which should flash its LED. If not specified, `flash` utilizes drive 8.

5.8.2 flash Example

Let the drive LED flash on drive 10:

```
cbmctrl lock
flash 10
cbmctrl unlock
```

5.9 morse

morse is a demo program. It uses the drive LED to output a text in morse code. *morse* works with 1541, 1570 or 1571 drives. A 1581 drive is *not* supported.

For Linux, *morse* is not installed automatically. You have to compile it yourself (found in `demo/morse/`) if you want to use it. For Windows, it is part of the binary distribution.

5.9.1 morse usage

Synopsis: `morse` *`[device]`*

The optional parameter *device* is the device number of the drive which should flash its LED. If not specified, `morse` utilizes drive 8.

5.9.2 morse Examples

Morse the text "SOS", "HELLO" and "YOU" (in this order) on drive 9.

```
cbmctrl lock
morse 9
cbmctrl command 9 U3:HELLO
cbmctrl command 9 U3:YOU
cbmctrl unlock
```

6 opencbm API

All communication between the user space applications and the kernel module is done with ioctl's. Since ioctl's are quite unportable and hardly provide any type-safety, there are a number of wrapper-functions along with a couple of convenience functions implemented in `libopencbm.a` (Linux) or `opencbm.dll` (Windows). The prototypes can be found in the header file `opencbm.h`.

6.1 Preprocessor macros

- #define IEC_DATA 0x01
- #define IEC_CLOCK 0x02
- #define IEC_ATN 0x04

These defines are used by the *cbm_iec_*()* functions. You will definitely need this if you intend to implement your own custom transfer routines. See the `libd64copy/libcbmcopy` source for more information.

6.2 Enumeration types

- enum cbm_device_type_e
  - cbm_dt_unknown
  - cbm_dt_1541
  - cbm_dt_1570
  - cbm_dt_1571
  - cbm_dt_1581
- enum cbm_cable_type_e
  - cbm_ct_unknown
  - cbm_ct_none
  - cbm_ct_xp1541

6.3 Generic types

- CBM_FILE

This type is used to take a handle to the CBM driver. Only use this type, as it hides the differences between Windows and Linux.

An invalid CBM_FILE has value CBM_FILE_INVALID.

6.4 Functions

(All functions except `cbm_driver_open()`: f must be a valid file descriptor)

6.4.1 Basic I/O

int cbm_driver_open(CBM_FILE *f, int port);

Opens the driver. `port` isn't used by now and should be 0. After successful completion, 0 is returned along with a valid CBM_FILE descriptor in `f`.

void cbm_driver_close(CBM_FILE f);

Closes the driver.

void cbm_lock(CBM_FILE f);

The equivalent to *cbmctrl lock*. Make sure the parallel port is kept locked even if the driver is closed with cbm_driver_close().

void cbm_unlock(CBM_FILE f);

The equivalent to *cbmctrl unlock*. Unlock the parallel port as soon is the driver is closed with cbm_driver_close().

int cbm_raw_read(CBM_FILE f, void *buf, size_t size);

Retrieve data after `cbm_talk();`. At most `size` bytes are read. Return value is the actual number of bytes read. $<$ indicates an error.

int cbm_raw_write(CBM_FILE f, const void *buf, size_t size);

Send data after `cbm_listen();`. At most `size` bytes are written, Return value is the actual number of bytes written. < indicates an error.

int cbm_listen(CBM_FILE f, __u_char dev, __u_char secadr);

Tell device *dev* to listen on secondary channel *secadr*. Return value is 0 on success, < 0 means error.

int cbm_talk(CBM_FILE f, __u_char dev, __u_char secadr);

Tell device *dev* to talk on secondary channel *secadr*. Return value is 0 on success, < 0 means error.

int cbm_open(CBM_FILE f, __u_char dev, __u_char secadr);

Prepare device *dev* for opening a file. This device listens for the file name after this call which is normally sent by a call to the *write()*-function followed by an *unlisten()* call. Return value 0 on success, < 0 means error.

int cbm_close(CBM_FILE f, __u_char dev, __u_char secadr);

Close file associated with secondary address *secadr* on device *dev*. Return value 0 on success, < 0 means error.

int cbm_unlisten(CBM_FILE f);

Send unlisten on bus. Return value 0 on success, < 0 means error.

int cbm_untalk(CBM_FILE f);

Send untalk on bus. Return value 0 on success, < 0 means error.

int cbm_get_eoi(CBM_FILE f);

Get EOI flag after bus read, return value is 0 with no EOI, otherwise 1. When EOI is set to 1, the active talker has nothing more to send.

int cbm_clear_eoi(CBM_FILE f);

Reset EOI flag. Return value 0 on success, < 0 means error.

int cbm_reset(CBM_FILE f);

Do a hardware reset on all connected devices. Control is returned after a 5 second delay.

6.4.2 Low-level port access

__u_char cbm_pp_read(CBM_FILE f);

Read byte from XP1541/XP1571 bus. No handshaking or such involved.

void cbm_pp_write(CBM_FILE f, __u_char c);

Write byte to XP1541/XP1571 bus. No handshaking or such involved.

int cbm_iec_poll(CBM_FILE f);

Read status of all bus lines. Return value is a combination of `IEC_ATN`, `IEC_CLOCK` and `IEC_DATA`.

int cbm_iec_get(CBM_FILE f, int line);

Get (logical) status of line *line*.

void cbm_iec_set(CBM_FILE f, int line);

Activate lines *line* (set to 0V). *line* can be one of or a combination with OR of any of IEC_DATA, IEC_CLOCK, IEC_ATN.

void cbm_iec_release(CBM_FILE f, int line);

Release lines *line* (set to 5V). *line* can be one of or a combination with OR of any of IEC_DATA, IEC_CLOCK, IEC_ATN.

void cbm_iec_setrelease(CBM_FILE f, int setline, int resetline);

Set lines *setline* (set to 0V) and release line *releaseline* (set to 5V) *setline* and *resetline* can each be one of or a combination with OR of any of IEC_DATA, IEC_CLOCK, IEC_ATN. If a line is part of both *setline* and *resetline*, the outcome is undefined.

int cbm_iec_wait(CBM_FILE f, int line, int state);

Experimental, do not use.

6.4.3 Helper functions

int cbm_upload(CBM_FILE f, __u_char dev, int adr, void *prog, int size);

Write *prog* into device *dev*'s memory space via a series of `"M-W"` commands.

int cbm_device_status(CBM_FILE f, __u_char drv, void *buf, int bufsize);

Read device status info *buf*, at most *bufsize* bytes are read. Returns *atoi(buf)*.

int cbm_exec_command(CBM_FILE f, __u_char drv, void *cmd, int len);

Execute command *cmd*. Returns number of bytes actually written. if *len* is 0, *cmd* is considered a 0-terminated string.

int cbm_identify(CBM_FILE f, __u_char drv, enum cbm_device_type_e *t, const char **type_str);

Tries to identify the device *drv*. The hardware type is returned in *t*, *type_str* contains a descriptive string which also includes the drives' operating system. Both *t* and *type_str* may be `NULL` in case the caller is not interrested in any of both values.

The return value is `0` if the device responded to the `"M-R"` command, even if it could not be identified, < 0 indicates error.

**int cbm_identify_xp1541(CBM_FILE f, __u_char drv, enum cbm_device_type_e *t1, enum cbm_cable_type_e *

Tries to identify the device *drv*. The hardware type is returned in *t1*, *t2* contains whether the drive has an parallel (XP1541) cable attached. Both *t1* and *t2* may be `NULL` in case the caller is not interrested in any of both values.

The return value is `0` if the device responded to the `"M-R"` command, even if it could not be identified, < 0 indicates error.

6.4.4 PetSCII functions

char cbm_petscii2ascii_c(char character);

Converts one character *character* from PetSCII to ASCII.

char cbm_ascii2petscii_c(char character);

Converts one character *character* from ASCII to PetSCII.

char * cbm_petscii2ascii(char *str);

Convert a null-terminated string *str* from PetSCII to ASCII.

char * cbm_ascii2petscii(char *str);

Convert a null-terminated string *str* from ASCII to PetSCII.

6.4.5 Parallel Burst functions

__u_char cbm_parallel_burst_read(CBM_FILE f);

Support function for mnib. Do not use.

void cbm_parallel_burst_write(CBM_FILE f, __u_char c);

Support function for mnib. Do not use.

int cbm_parallel_burst_read_track(CBM_FILE f, __u_char *buffer, unsigned int length);

Support function for mnib. Do not use.

int cbm_parallel_burst_write_track(CBM_FILE f, __u_char *buffer, unsigned int length);

Support function for mnib. Do not use.

6.4.6 libd64copy *TODO*

Not documented yet. See *libd64copy* and *d64copy* source.

Types and prototypes are defined in `d64copy.h`.

6.4.7 libcbmcopy *TODO*

Not documented yet. See *libcbmcopy* and *cbmcopy* source.

Types and prototypes are defined in `cbmcopy.h`.

7 Known bugs and problems

There are some known bugs in opencbm:

- *cbmcopy* is still known to have some protocol races, especially with 1581 drives; thus, it does not always work reliably.
- *cbmctrl detect* as well as *cbmcopy* and *d64copy* do not recognize the drive type if some custom ROM is used.
- Windows: If you have any other devices connected to your parallel port, you cannot use them as long as cbm4win is installed. In this case, either remove opencbm whenever you want to access that other device, or install opencbm with *instcbm –lock=no* and make sure to issue *cbmctrl lock* before accessing the drive, and *cbmctrl unlock* afterwards.
- Windows: No third party PCI parallel port card does work with opencbm on Windows currently; to say it with other words: there is no proof or positive report that any third party PCI parallel port card does or did work with opencbm on Windows. The exact failure reason is not known to date, but we are investigating further since that feature is a must, when integrated parallel ports were removed from mainstream mainboards in the future.

8 Misc

8.1 Credits

The fast format drive routine used by 'cbmformat' and the turbo and warp drive routines used in 'libd64copy' and 'libcbmcopy' are heavily based on Joe Forster/STAs Star Commander routines.

The XP1541 and XP1571 cables (C) by Joe Forster/STA. The original XE1541 cable (C) by Nicolas Welte and Wolfgang Moser The XA1541 cable (C) by Michael Klein and Nicolas Welte

8.2 Contributions

People who directly or indirectly contributed to opencbm (in no particular order):

- *Michael Klein* started the original cbm4linux work (which was a very big part)
- *Joe Forster/STA* made the Star Commander and supplied the source and about the X?1541 interfaces; who knows, without this work, opencbm might never have appeared at all.
- *Nicolas Welte* helped with the XA1541 and XM1541 interfaces and supplied a free factory-new 1571 mechanic for Michael
- *Andreas Boose & the VICE team* made VICE
- *André Fachat* made the xa 6502 crossassembler
- *Ullrich von Bassewitz* made the ca65 crossassembler
- *Wolfgang Moser* contributed *many* discussions, patches, and hardware whenever it was needed.
- *Spiro Trikaliotis* with discussions, lots of fixes and doing an overall great review while porting the driver to "other" operating systems ;-)

And anyone else who sent patches, suggestions, praises & flames!

8.3 Feedback

Feel free to drop a note if you have ideas, patches etc. or if you just want to tell how happy you are with this program ;-)

Have fun,

The opencbm team.