The EasySDK Guide

by Thomas 'skoe' Giesel
skoe@directbox.com

Table of contents

Last change: 26/01/11

1 Hardware Description

An EasyFlash cartridge consists of two flash memory chips, called LOROM chip and HIROM chip. Each of them has a size of 512 KiB. There may be cartridge versions where these both logical chips are merged to one physical chip.

Only 8 KiB of memory per chip can be seen in the address space of the C64 at once. Therefore EasyFlash supports banking. When each chip has a size of 512 KiB, there are 512 / 8 = 64 banks. The EasyFlash cartridge allows software to select the active bank.

The mapping of the chips into the C64 memory is controlled by two lines of the Expansion Port: /GAME and /EXROM. The "/" means that their meaning is inverted, i.e. 0 or low means active.

The possible states of these lines and their meaning is listed in Table 1.

/GAME	/EXROM	Comment
1	1	Cartridge memory invisible ("Invisible Mode")
1	0	8 KiB from Chip 0 at $8000 ("8K Mode")
0	0	8 KiB from Chip 0 at $8000, 8 KiB from Chip 1 at $A000 ("16K Mode")
0	1	8 KiB from Chip 0 at $8000, 8 KiB from Chip 1 at $E000 ("Ultimax Mode")

Table 1: States of /GAME and /EXROM lines

These lines can be controlled by software, we'll come back to this topic soon.

The resulting memory model in *8k Mode / 16K Mode / Ultimax Mode* is shown in Fig. 1.

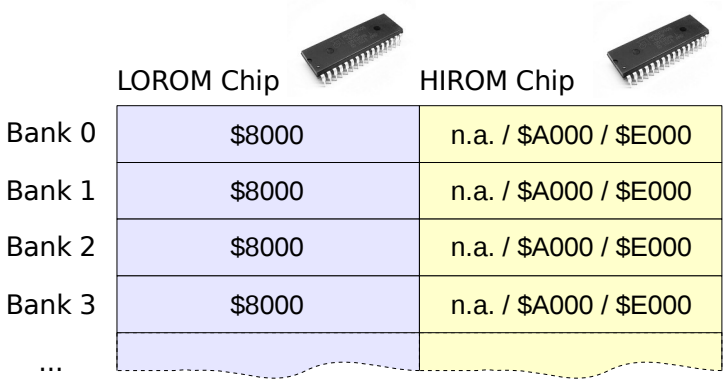

Fig. 1: EasyFlash memory model

If the Boot switch is in position "Boot" (or jumper open) and the computer is reset, EasyFlash is started normally: The Ultimax memory configuration is set, bank 0 selected. The CPU starts at the reset vector at $FFFC. As in Ultimax mode the HIROM chip is banked in at $E000, the flash must contain a valid reset vector and some start-up code. This is described in a separate chapter.

If the Boot jumper is in position "Disable" (or jumper closed), EasyFlash starts with cartridge memory hidden. However, software like EasyProg can write to the flash memory because it can override the jumper setting.

The software can chose the active bank and set the memory location by using two I/O registers. They are described in the following chapters.

Additionally an EasyFlash cartridge has 256 Bytes of RAM. This memory is always visible. It can be used to save small portions of code or data in a way which is unlikely to interfere with other software.

1.1 Register $DE00 – EasyFlash Bank

This is a write-only register located at $DE00. The active flash memory bank is chosen using this register. Basically it selects the state of the higher address lines. The bank selection can be made regardless if the flash memory is actually visible according to the memory configuration or not. The selected bank is valid for both chips, LOROM and HIROM.

Bit	7	6	5	4	3	2	1	0
Meaning	0	0	Bank number					

Table 2: Register $DE00

The value after reset is $00. The bits 6 and 7 should always remain 0.

1.2 Register $DE02 – EasyFlash Control

This is a write-only register located at $DE02. The software can control the memory configuration of the cartridge using this register. Additionally it can be used to set the state of the status LED.

Bit	7	6	5	4	3	2	1	0
Meaning	L	0	0	0	0	M	X	G

Table 3: Register $DE02

The value after reset is $00.

Position	**Name**	**Comment**
7	L	Status LED, 1 = on
6..3	0	reserved, must be 0
2	M	GAME mode, 1 = controlled by bit G, 0 = from jumper "boot"
1	X	EXROM state, 0 = /EXROM high
0	G	GAME state if M = 1, 0 = /GAME high

Table 4: Bits of $DE02

Following memory configurations are possible:

MXG	**Type**
000	GAME from jumper, EXROM high (i.e. Ultimax or Off)
001	Reserved, don't use this
010	GAME from jumper, EXROM low (i.e. 16K or 8K)
011	Reserved, don't use this
100	Cartridge ROM off (RAM at $DF00 still available)
101	Ultimax (Low bank at $8000, high bank at $e000)
110	8k Cartridge (Low bank at $8000)
111	16k cartridge (Low bank at $8000, high bank at $a000)

Table 5: Configuration of cartridge memory maps

Most likely the software is only interested in setting combinations with M = 1. Of course the usual memory configurations selected with the 6510 register $01 do also apply.

1.3 RAM at $DF00

From $DF00 to $DFFF there are 256 bytes of RAM. This memory is not part of the normal 64 KiB of internal C64 RAM but part of the I/O memory space.

2 Conventions

Conventions can make the life easier and look professional. So let's invent some.

2.1 Address Format Convention

Addresses of EasyFlash have a special format in this document and related tools:

BB:C:FFFF

BB	Bank number, currently supported range 00..3F
C	Chip number, 0 for LOROM chip or 1 for HIROM chip
FFFF	Offset in this bank and chip, 0000..1FFF

Table 6: Address format convention

The address 00:1:1FFC means bank 0, HIROM chip, offset 0x1FFC. This is the address which has to contain a reset vector to make the cartridge work.

2.2 Scan the Keyboard

When a cartridge is plugged into the C64, often the user has no other way to get to the BASIC prompt than unplugging the cartridge.

EasyFlash has a switch or a jumper to avoid booting. However, let's make it even easier for our users:

When you implement start-up code for EasyFlash, scan the keyboard as early as possible. If one of the keys *<Run/Stop>, <Commodore> or <Q>* is held down, make the cartridge invisible and call the Kernal's reset vector. Remember that you can hide (“kill”) the cartridge by writing $04 to $DE02.

3 Supported cartridge types

Different types of cartridge images can be written to and started from an EasyFlash cartridge. Let's have find out how these work.

3.1 Normal 8K Cartridges

Normal 8K cartridges consist of up to 8 KiB of ROM visible at $8000 (LOROM). This type of cartridges pulls down /EXROM and leaves /GAME high.

The memory at $8000 contains some magic bytes. The C64 Kernal detects these bytes in the reset code. If such a cartridge is found, it gets started with JMP($8000).

EasyFlash cartridges always start in Ultimax mode. Therefore EasyProg adds start-up code automatically when flashing 8K cartridges.

3.2 Normal 16K Cartridges

Normal 16K cartridges consist of up to 16 KiB of ROM. 8 KiB are visible at $8000 (LOROM) and 8 KiB at $A000 (HIROM). This type of cartridges pulls down /EXROM and /GAME.

The memory at $8000 contains some magic bytes. The C64 Kernal detects these bytes in the reset code. If such a cartridge is found, it gets started with JMP($8000).

EasyFlash cartridges always start in Ultimax mode. Therefore EasyProg adds start-up code automatically when flashing 8K cartridges.

3.3 Ultimax Cartridges

Ultimax cartridges consist of up to 16 KiB of ROM. 8 KiB are visible at $8000 (LOROM) and 8 KiB at $E000 (HIROM). This type of cartridges pulls down /GAME and leaves /EXROM high.

The memory at $e000 contains the reset vector. When the CPU is reset, the execution starts at the address pointed to by this vector. This kind of cartridges must initialize the hardware carefully, because there is no kernal doing this for you.

Even for Ultimax Cartridges EasyProg adds start-up code automatically when flashing. The reason is that this start-up code scans the keyboard, see chapter 2.2.

3.4 Ocean Cartridges

Ocean cartridges are banked cartridges. The bank number is written to $DE00, like it is done on EasyFlash.

Real Ocean cartridges do always set bit 7 when they write to $DE00. This bit is ignored by the current hardware version of EasyFlash and most likely by the emulators.

Ocean cartridges with up to 256 KiB use following banking scheme:

Up to 16 Banks are put to LOROM. This means that banks number 0..15 can be found at LOROM. If needed, banks 16..31 are put to HIROM. These cartridges pull down /EXROM and /GAME, so 16 KiB of ROM are mapped into the C64 memory at once. 8 KiB at $8000 and 8 KiB at $A000.

When banks 0..15 are used, which are read from LOROM, the content of HIROM is not used. When banks 16..31 are used, which are read at HIROM, the content of LOROM is not used. This means although 16 KiB are banked in at once, one 8 KiB of each bank are actually used.

A CRT image only contains the banks actually used.

Example 1: A 128 KiB Ocean cartridge looks like this (**'P'** = Data, **'/'** = undefined / don't care, **'-'** = nothing):

```
BANK  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F

LOROM P P P P P P P P P P P P P P P P - - - - - - - - - - - - - - - -
HIROM / / / / / / / / / / / / / / / / - - - - - - - - - - - - - - - -
```

Example 2: A 256 KiB Ocean cartridge looks like this:

```
BANK  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F

LOROM P P P P P P P P P P P P P P P P / / / / / / / / / / / / / / / /
HIROM / / / / / / / / / / / / / / / / P P P P P P P P P P P P P P P P
```

There is a completely different type of Ocean cartridge, the 512 KiB type (Terminator II):

This type does also use $DE00 for banking. It uses 64 banks of 8 KiB each. All banks are read from 8 KiB of ROM visible at $8000 (LOROM). This type of cartridges pulls down /EXROM and leaves /GAME high.

Example 3: A 512 KiB Ocean cartridge looks like this:

```
BANK  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 ... 3 3
      0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E 0 ... E F

LOROM P P P P P P P P P P P P P P P P P P P P P P P P P P P P P P ... P P
```

Ocean cartridge have the advantage to be also compatible to older emulators. On the other hand the format isn't very intuitive.

EasyFlash cartridges always start in Ultimax mode. Therefore EasyProg adds start-up code automatically when flashing Ocean cartridges.

3.5 Native EasyFlash Cartridges

Native EasyFlash Cartridges can use the full flash memory of 1 MiB. They can use any kind of banking which is supported by the EasyFlash hardware in any way they want.

EasySDK comes with some tools and code snippets to create, write and read EasyFlash cartridges.

As already mentioned, EasyFlash cartridges always start in Ultimax mode. Therefore there is a small boot code required at the end of the HIROM flash chip on bank 0 (00:1:1xxx). This start-up code is executed directly after a CPU reset.

The start-up code has to:

- Provide the reset vector
- Scan the keyboard
- Set up $DE02 and start the stuff

A reference implementation for the start-up code can be found in *examples/banking-test*.

If a native EasyFlash CRT wants to write to the flash memory, it may contain a piece of code called EasyAPI. This is described in chapter 4.

The memory area from 00:01:1800 to 00:01:1BFF is reserved for EasyAPI. So the startup code and data can occupy e.g. the area 00:01:1C00 to 00:01:1FFF.

3.6 EasyFlash xbank Cartridges

The EasyFlash xbank cartridge format is a multi-bank cartridge format which can be placed at any bank in the flash, though all the banks must be continuous. This makes it possible to combine several xbank cartridges into one cartridge. The format is similar to the EasyFlash cartridge format.

The magic cartridge id is 33 ($21).

The main difference is that it can be placed on any bank, unlike EasyFlash or e.g. Ocean Type 1 cartridges.

The GAME and EXROM configuration in the CRT header will be used to determine which mode should be set at start-up. This is done by the start-up code. This code will also write the right start bank to $DE00 and start the cartridge using the CPU Reset Vector.

The start bank will also be stored in $DF00 and the program has to care itself for the addition of the relative bank and the offset before writing it into the banking register.

EasyProg adds start-up code automatically when flashing an xbank cartridge directly.

Currently EasyProg does not automatically replace/patch EasyAPI for xbank cartridges.

4 EasyAPI

Applications can write to EasyFlash cartridges using a small library which is called EasyAPI. This application programming interface supports to erase blocks of flash memory and write data to flash. Additionally it can be used to read from flash memory. EasyAPI can be seen as a flash chip driver, similar to drivers for your sound card or video card.

EasyAPI can be part of your cartridge software. If you put it at a special position, EasyProg (the tool to write CRT images to cartridges) will automatically update the EasyAPI code to the latest version when your cartridge is written to flash memory. This is **strongly recommended**, because there are different EasyFlash hardware versions already which need an updated EAPI.

EasyAPI can also be used in programs running from disk. It can be loaded from a file and used to write data to flash memory. That's the way EasyProg does it.

Note that EasyAPI does not support to write real files to the flash and to a EasyFS directory structure. Writing is done on a lower logical level only, by directly addressing flash memory.

4.1.1 Using EasyAPI in Cartridges

There is one very important feature for EasyAPI: When a CRT image which contains the “EAPI” signature at the right place is written to a cartridge by EasyProg, the EasyAPI memory area is updated with the latest version of the EasyAPI code. The API of this code will remain compatible, but the new code may support new Flash chip types or bug fixes.

This situation is solved by the replace mechanism: When an updated version of EasyProg is used to write the old CRT image, the EasyAPI is automatically replaced with an updated version.

To add the EasyAPI code to your self-implemented CRT images, include the binary (e.g. “`eapi-am29f040-03`”) at 00:01:1800 into your CRT image. Remember to reserve up to 768 bytes of memory for future versions of EasyAPI.

4.1.2 Using EasyAPI in ordinary programs

EasyAPI can also be used in ordinary programs which are started e.g. from a disk. They must search for a file called “**eapi-????????-??**” (note the number of wildcards '?') on the current drive and load it to any RAM area in the range $0200...$7FFF or $C000...$CFFF.

Programmers should prefer not to link EasyAPI directly into their programs, otherwise it cannot be replaced by newer versions of the file later.

The current version of EasyAPI supports the flash chip Am29F040. The actual file name is:

“eapi-am29f040-03”

The following chapters explain the entries to the jump table.

4.1.3 EasyAPI Memory Usage

The functions of EasyAPI do not pollute the C64 RAM. All data they need is stored in the EasyFlash RAM from $DF80 to $DFFF. One special API function named *EAPIInit* must be called first to initialize the jump tables. *EAPIInit* temporarily uses the zeropage locations $4b and $4c, but it backs them up and restores them before the function returns.

When you use the RAM at $df80 to $dfff for something else, remember to call *EAPIInit* again before using other EasyAPI functions.

4.1.4 EasyAPI Code Position

As mentioned already, you can load EasyAPI to any RAM area in the range $0200...$7FFF or $C000...$CFFF. The first byte must be page aligned, EasyAPI may be up to 768 bytes of size.

If you use EasyAPI from a cartridge, you must copy it from ROM to any RAM area as described above. The reason is that the code uses bank switching and would bank itself out otherwise.

4.1.5 EasyAPI Signature

Offset	Length	Comment
0	4	EasyAPI signature, always $65 $61 $70 $69 (“EAPI”)

This signature is only used to show the existence of EasyAPI. If the tool EasyProg finds this signature, it can replace this memory area 00:1:1800 to 00:1:1BFF with a newer version of EasyAPI.

4.1.6 EasyAPI Version String

Offset	Length	Comment
4	16	EasyAPI version string, 0-terminated PETSCII

This string contains the version of EasyAPI. It is for informational purpose only.

4.1.7 EAPIInit

Must be called with JSR <load_address> + 20 where <load_address> is the address in RAM where you copied EasyAPI to.

Read Manufacturer ID and Device ID from the flash chip(s) and check if this chip is supported by this driver. Prepare our private RAM for the other functions of the driver. When this function returns, EasyFlash will be configured to bank in the ROM area at $8000..$bfff.

This function uses SEI, it restores all Flags except C before it returns. Do not call it with D-flag set. $01 must enable both ROM areas.

Parameters

-

Return

C set: Flash chip not supported by this driver

clear: Flash chip supported by this driver

If C ist clear:

A Device ID

X Manufacturer ID

Y Number of physical banks (64 for Am29F040)

Changes

All registers are changed

4.1.8 EAPIWriteFlash – $DF80

To be called with JSR jmpTable + 0 = $df80.

Write a byte to the given address. The address must be as seen in Ultimax mode, i.e. do not use the base addresses $8000 or $a000 but $8000 or $e000.

When writing to flash memory only bits containing a '1' can be changed to contain a '0'. Trying to change memory bits from '0' to '1' will result in an error. You must erase a memory block to get '1' bits.

This function uses SEI, it restores all Flags except C before it returns. Do not call it with D-flag set. $01 must enable the affected ROM area. It can only be used after having called EAPIInit.

Parameters

A value

XY address (X = low), $8xxx/$9xxx or $Exxx/$Fxxx

Return

C set: Error
clear: Okay

Changes

Z,N <- value

4.1.9 EAPIEraseSector – $DF83

To be called with JSR jmpTable + 3 = $df83.

Erase the sector at the given address. The bank number currently set and the address together must point to the first byte of a 64 kByte sector.

When a sector is erased, all bits of the 64 KiB area are set to '1'. This means that 8 banks with 8 KiB each will be erased, all of them either in the LOROM chip when $8000 is used or in the HIROM chip when $e000 is used.

This function uses SEI, it restores all flags except C before it returns. Do not call it with D-flag set. $01 must enable the affected ROM area.

; It can only be used after having called EAPIInit.

Parameters

A bank
Y base address (high byte), $80 for LOROM, $a0 or $e0 for HIROM

Return

C set: Error
clear: Okay

Changes

Z,N <- bank

4.1.10 EAPISetBank – $DF86

To be called with JSR jmpTable + 6 = $df86.

Set the bank. This will take effect immediately for cartridge read access and will be used for the next write or read command.

This function can only be used after having called EAPIInit.

Parameters

A bank number

Return

-

Changes

Z,N <- bank

4.1.11 EAPIGetBank – $DF89

To be called with JSR jmpTable + 9 = $df89.

Get the selected bank which has been set with EAPISetBank. Note that the current bank number can not be read back using the hardware register $de00 directly, this function uses a mirror of that register in RAM.

This function can only be used after having called EAPIInit.

Parameters

-

Return

A bank number

Changes

Z,N <- bank

4.1.12 EAPISetPtr – $DF8C

To be called with JSR jmpTable + 12 = $df8c.

Set the pointer for EAPIReadFlashInc/EAPIWriteFlashInc.

This function can only be used after having called EAPIInit.

Parameters

A bank mode, where to continue at the end of a bank
$D0: 00:0:1FFF → 00:1:0000, 00:1:1FFF=>01:0:1FFF (LHLH...)
$B0: 00:0:1FFF → 01:0:0000 (LLLL...)
$D4: 00:1:1FFF → 01:1:0000 (HHHH...)

XY address (X = low) address must be in range $8000-$bfff

Return

-

Changes

-

4.1.13 EAPISetLen – $DF8F

To be called with JSR jmpTable + 15 = $df8f.

Set the number of bytes to be read with EAPIReadFlashInc.

This function can only be used after having called EAPIInit.

Parameters

XYA length, 24 bits (X = low, Y = med, A = high)

Return

-

Changes

-

4.1.14 EAPIReadFlashInc – $DF92

To be called with JSR jmpTable + 18 = $df92.

Read a byte from the current pointer from EasyFlash flash memory. Increment the pointer according to the current bank wrap strategy. Pointer and wrap strategy have been set by a call to EAPISetPtr.

The number of bytes to be read may be set by calling EAPISetLen. EOF will be set if the length is zero, otherwise it will be decremented. Even when EOF is delivered a new byte has been read and the pointer incremented. This means the use of EAPISetLen is optional.

This function can only be used after having called EAPIInit.

Parameters

-

Return

A value

C set if EOF

Changes

Z,N <- value

4.1.15 EAPIWriteFlashInc – $DF95

To be called with JSR jmpTable + 21 = $df95.

Write a byte to the current pointer to EasyFlash flash memory. Increment the pointer according to the current bank wrap strategy. Pointer and wrap strategy have been set by a call to EAPISetPtr.

In case of an error the position is not inc'ed.

This function can only be used after having called EAPIInit.

Parameters

A value

Return

C set: Error
clear: Okay

Changes

Z,N <- value

Appendix A Easy File System (EasyFS)

If you want to store files onto an EasyFlash cartridges, you have to implement a kind of file system. EasyFS is a file system which is used e.g. by the well known EasyLoader.

An example memory layout of an cartridge using EasyFS is shown in Fig. 2.

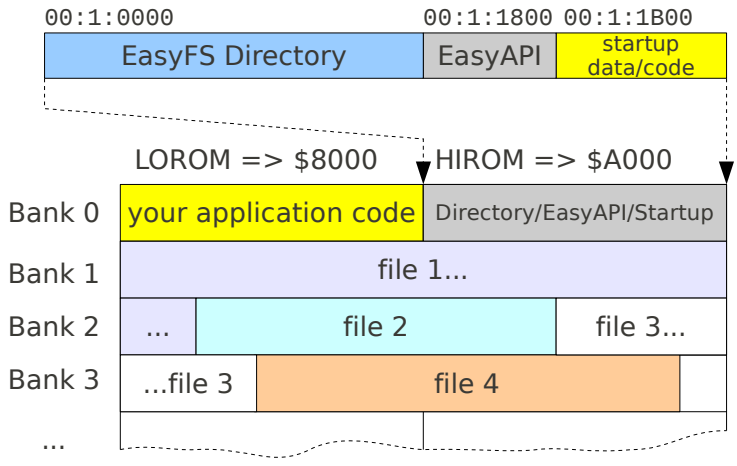

Fig. 2: An EasyFS cartridge

Cartridges using EasyFS always use the 16 KiB cartridge mode, which is configured by the start-up code by setting /GAME and /EXROM active (low). All explanations in this chapter refer to this configuration.

If your cartridge makes use of the Easy File System (EasyFS), it contains a directory structure. This structure resides at 00:1:0000. This means it is visible at $A000 in 16 KiB mode when bank 0 is selected.

A directory shall not contain more than 255 entries (excluding end mark). We will see that each entry has a size of 24 Bytes, so the whole directory can have a size of up to $1800 Bytes. Behind the directory there is space of $0800 bytes for EasyAPI and the start-up code and data.

Remember that HIROM is banked in in Ultimax mode at $E000 after a reset. Therefore the start-up code contains the reset vector.

The format of EasyFS directory entries is as follows:

Field	Type	Comment
Name	16 bytes	File name, 16 characters PETSCII (?), 0-padded
Flags	1 byte	File type and some flags
Bank	1 byte	Bank where the file starts, 0..63
Bank High	1 byte	Reserved for high byte of bank, currently always 0
Offset	2 bytes	File start offset in this bank 0x0000..0x3FFF, little endian
Size	3 bytes	File size, little endian

The flags field has following structure:

Bit	7	6	5	4	3	2	1	0
Meaning	H	R	R	Type				

The bit *H* means hidden. If this is 1, this file is not intended to be seen by a user, a file browser should not show it.

`Reserved bits marked with R must be set to 1.`

Table 7 shows the possible values for *Type*.

Value	Type
$00	`Files with this type are marked as invalid, they must be skipped. Note that the flags of this file may be not 0.`
$01	`Normal PRG file with 2 bytes start address`
...	
$10	Normal 8 kByte cartridge (0x8000..0x9FFF)
$11	Normal 16 kByte cartridge (0x8000..0x9FFF, 0xA000..0xBFFF)
$12	Normal Ultimax cartridge (0x8000..0x9FFF, 0xE000..0xFFFF)
$13	Normal Ultimax cartridge, first bank not used (0xE000..0xFFFF)
...	
$1F	End of directory. This entry is only a terminator.

Table 7: EasyFS file types

Note that erased flash memory does have the value $FF, so all bits are *one*. This means that an entry located in erased flash has the type 0x1f naturally.

Furthermore, when a flash is written, bits can only turn from *one* to *zero*. It is not possible to change a bit from *zero* to *one* when writing to flash memory. That's why the file type $00 marks a deleted or invalid file. $00 can be written regardless from the old file type by only changing *ones* to *zeros*.

The meaning of the file types $10 to $13 will be described on page 17.

Appendix B Hybrid cartridges

An image may also contain one or more normal 8 kByte, 16 kByte or Ultimax sub-modules. These have to be placed bank-aligned, because they are not loaded, but banked in and started. This special feature may be useful for something like CRT collections.

That is the reason a directory may also have entries which refer to sub-cartridges.

Fig. 3 shows an example hybrid cartridge which contains a 16K sub-cartridge.

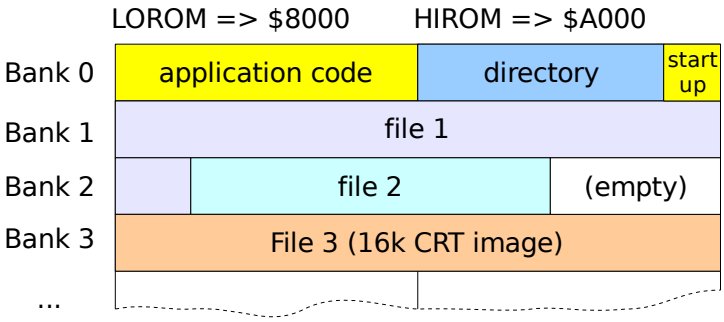

Fig. 3: Hybrid cartridge containing a normal 16K cartridge