The EasySDK Guide

Table of contents

1 Hardware Description

An EasyFlash cartridge consists of two flash memory chips, called Chip 0 and Chip 1. In the current hardware version each of them has a size of 512 KiB.

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.

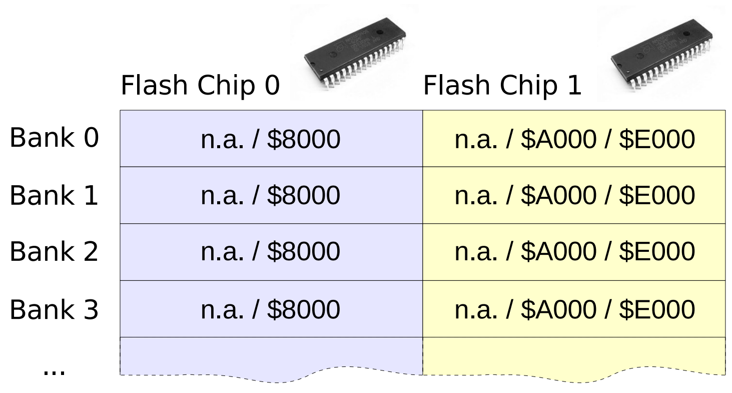
The mapping of the chips into the C64 memory is controlled by two lines of the Expansion Port: /GAME and /EXROM. The “/” means that their meaning is inverted, i.e. 0 or low means active.

The possible states of these lines and their meaning is as follows:

/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”)

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

Let's have a look at the 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 flash chip 1 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.

The software can chose the active bank and set the memory location 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 read-only register located at $DE00. The flash memory bank with is currently active 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.

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

The value after reset is $00.

1.2 Register $DE02 – EasyFlash Control

This is a read-only register located at $DE02. The software can control the memory configuration of the cartridges 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

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

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 memory off`
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)`

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.

1.4 Address format convention

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

```
BB:C:FFFF
```

BB	Bank number, currently 00..3F
C	Chip number, 0 for low flash chip or 1 for high flash chip

FFFF	Offset in this bank and chip, 0000..1FFF

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

2 EasyFS Cartridges

2.1 Introduction

EasyFS means Easy File System. This is a simple ROM file system which simplifies implementing software for EasyFlash.

EasySDK comes with some tools and code snippets to create and read EasyFS.

Of course developers are free to develop cartridges which do not use EasyFS by directly accessing the I/O registers described in this document.

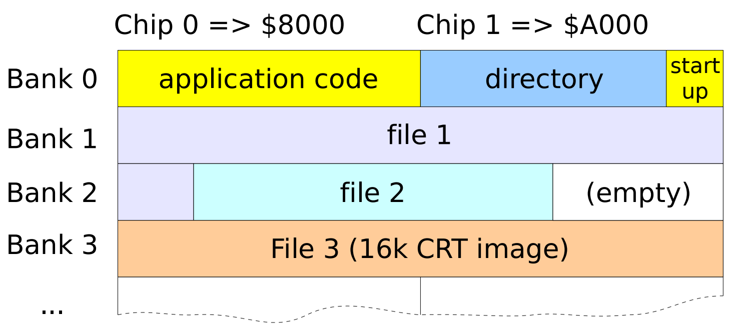
As already mentioned earlier, EasyFlash cartridges always start in Ultimax mode. Therefore there is a small boot code at the end of the high flash chip on bank 0, i.e. at 00:1:FFxx. Cartridges using EasyFS always use the 16 KiB cartridge mode, which is configured by the start-up code by setting /GAME and /EXROM active (low). All explanations in this chapter refer to this configuration.

If your cartridge makes use of the EasySDK 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.

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

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

This is the ROM (Flash) memory layout of an EasySDK cartridge using EasyFS:



Remember that chip 1 is banked in in Ultimax mode at $E000 after a reset. Therefore the start-up code contains the reset vector.

2.2 Format of EasyFS Directory Entries

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. 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.`

Following *Type*s are defined:

Value	**Type**
0x00	`Files with this type are marked as invalid, they must be skipped. Note that the flags of this file may be not be 0.`
0x01	`Normal PRG file with 2 bytes start address`
...	
0x10	Normal 8 kByte cartridge (0x8000..0x9FFF)
0x11	Normal 16 kByte cartridge (0x8000..0x9FFF, 0xA000..0xBFFF)
0x12	Normal Ultimax cartridge (0x8000..0x9FFF, 0xE000..0xFFFF)
0x13	Normal Ultimax cartridge, first bank not used (0xE000..0xFFFF)
...	
0x1F	End of directory. This entry is only a terminator.

2.3 Start-Up Code

Todo: Vorbereiteter Start-Up-Code kopiert Code zum Zugriff auf EasyFS nach $CF00, schaltet in den 16 KiB-Modus und ruft Code an $8000 auf.

3 Flash Write API

Todo: EasyProg soll automatisch auf die letzte Kachel ein paar Funktionen legen, die das Schreiben in den Flash ermöglichen. Sinnvoll für Spielstände o.ä.?

Aber nur, wenn die Kachel nicht ohnehin benutzt wird.

Dieses API ist nicht im eigentlichen CRT-Image enthalten, damit es in zukünftigen Versionen mit anderen Flash-Typen auch noch funktioniert.

Offen: Und wie geht das in Emulatoren?