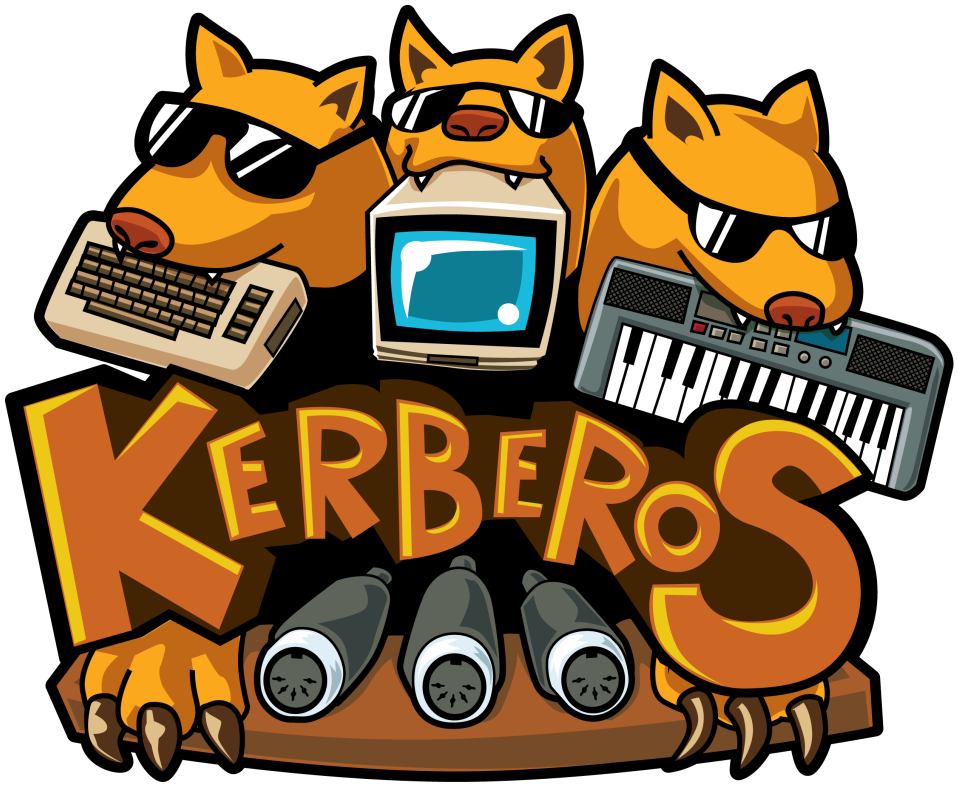

Kerberos C64/C128 MIDI Flash Interface Developer's Guide

FRANK BUSS

MAKING NEW TECHNOLOGY WORK

table of contents

How to write your own programs for Kerberos

C development environment

A good start is to clone the Kerberos repository (https://github.com/FrankBuss/kerberos/) and compile the synthesizer program (in c64/src/synthesizer.c), which you can use as a starting point for your own programs. It receives note-on/note-off events on MIDI channel 1-4 and uses the SID to play it.

If you work on Windows, first you need to install Cygwin (https://www.cygwin.com), then cc65 (tested with the Windows snapshop at http://sourceforge.net/projects/cc65/files/cc65-snapshot-win64.zip/download , version "cc65 V2.14 - Git d75f9c2", because it needs some special linker features) and add the cc65 to the system path. Then in a Cygwin shell, change to the kerberos/c64 path, run "make" and you should get synthesizer.prg, which is the C64 PRG file for the program above.

You can even test the program in VICE. On Windows enable the MIDI interface: Open "Settings->Cartridge/IO Settings->MIDI Settings", then enable "Enable MIDI emulation" and choose "Namesoft" as "MIDI type" and your MIDI-In device. You could loopback from your MIDI-Out device or send it data from another PC or Mac. In the synthesizer.c program you need to delete the midiIrqNmiTest, if you don't use the latest VICE emulator from the Sourceforge source repository, because there was a bug in VICE 2.4 regarding the send IRQ, which will be available in the new VICE 2.6 version.

Using the SRAM

To use the 128 k SRAM memory extension on the cartridge, you can select the bank (256 bytes block) with the RAM_ADDRESS_EXTENSION and RAM_ADDRESS_EXTENSION2 register (see c64/src/regs.h and the ramSetBank helper function in the menu source code how to use it) and then use $df00-$dfff to write and read to the selected bank. This is possible in BASIC, too.

BASIC example

BASIC is too slow to receive MIDI signals, but you could use the assembler MIDI functions and add some functions to use it from BASIC instead of C. But sending notes is possible with BASIC. The code below is an example which sends note-on and note-off events to the first MIDI channel. You can test it in VICE, too, with "DATEL/Siel/JMS" as the MIDI emulation.

```
10 PRINT"MIDI NOTE-ON/OFF TEST"
20 GOSUB 1000
30 V = 100
40 FOR N = 60 TO 100
50 PRINT"NOTE:";N
60 GOSUB 3000
70 FOR I = 0 TO 200:NEXT
80 GOSUB 4000
90 NEXT
100 GOTO40
1000 REM MIDI INIT
```

```
1010 REM CONTROL REGISTER
1020 CR = 56836
1030 REM STATUS REGISTER
1040 SR = 56838
1050 REM TRANSMIT REGISTER
1060 TR = 56837
1070 REM RECEIVE REGISTER
1080 RR = 56839
1090 REM RESET
1100 POKE CR, 3
1110 REM ENABLE
1120 POKE CR, 22
1130 RETURN
2000 REM SEND BYTE IN B
2010 POKE TR, B
2020 REM WAIT UNTIL SENT
2030 IF PEEK(SR) AND 2 = 0 THEN GOTO 2030
2040 RETURN
3000 REM NOTE-ON, N = NOTE, V = VEL.
3010 B = 144:GOSUB 2000
3020 B = N: GOSUB 2000
3030 B = V: GOSUB 2000
3040 RETURN
4000 REM NOTE-OFF, N = NOTE
4010 B = 128:GOSUB 2000
4020 B = N: GOSUB 2000
4030 B = 0: GOSUB 2000
4040 RETURN
```

Kerberos App

The PC/Mac program, Kerberos App, is a Qt program and in the qt directory in the github repository. You can install the latest Qt Creator, if you want to compile it yourself or modify it. There are build-scripts to build and create the release packages for the systems (build-*.cmd/sh, with *=system name).

Flash and SRAM memory maps, and MIDI transfer format

Flash memory map

Memory map of the 2 MB integrated flash memory.

Start	End	Description
00.0000	00.7fff	32 k area for menu system, starts after reset as a module with CBM ID at $8000 (= flash address 0)
00.8000	00.9fff	8 k cartridge disk implementation, running from $8000 as ROM
00.a000	00.afff	4 k for flash quick test
00.b000	00.bfff	4 k settings
00.c000	00.dfff	8 k custom BASIC
00.e000	00.ffff	8 k custom KERNAL
01.0000	03.ffff	3 x 64 k 1st cartridge disk
04.0000	06.ffff	3 x 64 k 2nd cartridge disk
07.0000	0f.ffff	9 x 64 k slots to store 9 PRG-files
10.0000	1f.ffff	1 mb EasyFlash CRT area, or 16 additional PRG slots

Slot memory map

Memory map for one of the 25 slots.

$00-$0f: ID to indicate that the slot contains a valid PRG file: ASCII „KERBEROS PRGSLOT“
$10-$2f: ASCIIZ filename (max 31 bytes without the last zero), filled with zeros
$30: control byte
- bit 0: 1 = start program in C128 mode, 0 = start in C64 mode
- bit 1: 1 = copy BASIC replacement to SRAM before starting the program
- bit 2: 1 = copy KERNAL replacement to SRAM before starting the program
- bit 3: 1 = use global MIDI settings
- bit 4: 1 = enable cartridge disk before starting the program

$31-$38: unused
$39-$3f: values for registers $de39 - $de3f before starting the program
$40-$41: program load address
$42-$43: program start address (0 = BASIC RUN)
$44-$45: program length

$46-$ff: unused
$100-x : PRG data (without load address), „program length“ number of bytes

SRAM memory map

Memory map for the SRAM while the menu system is running and when starting with the cartridge disk code or one of the ROM as RAM functions for BASIC and KERNAL replacement. Otherwise the full SRAM is free for applications.

$0.0000-$0.00ff: cartridge disk trampoline code
$0.0100-$0.01ff: backup for zeropage while in cartridge disk code
$0.0200-$0.02ff: backup for other memory while in cartridge disk code
$0.0300-$0.03ff: RAM backup for the diskbuf
$0.0400-$0.7fff: unused
$0.8000-$0.9fff: cartridge disk implementation (with RAM as ROM function mapped)
$0.a000-$0.bfff: BASIC replacement
$0.c000-$0.dfff: unused
$0.e000-$0.ffff: KERNAL replacement
$1.0000-$1.ffff: current header and program to start, copied from slot or transfered by MIDI

MIDI transfer format

All data transfer is encoded in note-off messages:

$8n, n:
bit 3: 1=start of data transfer, 0=data bytes
bit 2: 1=two data bytes, 0=one data byte (second data byte of the note-off message is ignored)
bit 1: bit 7 of first data byte
bit 0: bit 7 of second data byte (if sent)

start of data transfer, with two data bytes ($8c)
first byte: tag ID. If bit 7 is set, then there is no data and the second byte is an optional data byte.
second byte: size-1 (0 means 1 data byte … $ff means $100 data bytes)
last byte in the next data messages: CRC8 checksum (calculated over tag, size and data)

Tags: see midi_commands.h header file