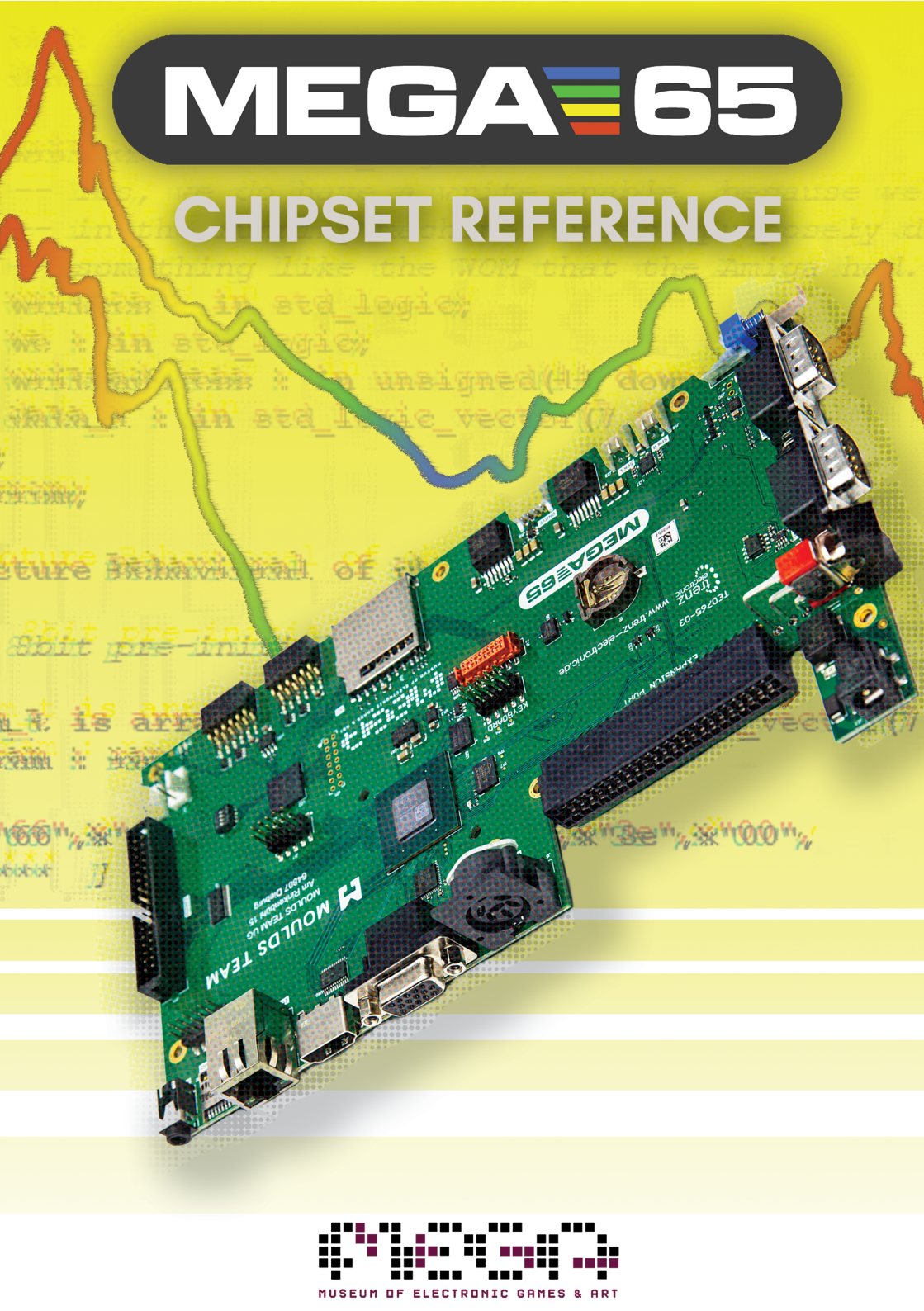
MEGA 65

CHIPSET REFERENCE



MUSEUM OF ELECTRONIC GAMES & ART

MEGA65 TEAM

Assoc. Prof. Paul Gardner-Stephen
(highlander)
Founder
Software and virtual hardware architect
Spokesman and lead scientist

Martin Streit
(seriously)
Video and photo production
Tax and organization
Social media

Dan Sanderson
(dddaaannn)
Media and documentation
MEGA65.ROM

Dr. Edilbert Kirk
(Bit Shifter)
MEGA65.ROM
Manual and tools

Gábor Lénárt
(LGB)
Emulator

Farai Aschwanden
(Tayger)
Filehost and tools
Financial advisory

Falk Rehwagen
(bluewaysw)
GEOS

Robert Steffens
(kibo)
Network technology
Core bug hunting

Detlef Hastik
(deft)
Co-founder
General manager
Marketing and sales

Oliver Graf
(lydon)
Release management
VHDL and platform enhancements

Antti Lukats
(antti-brain)
Host hardware design and production

Dieter Penner
(doubleflash)
Host hardware support

Mirko H.
(sy2002)
Additional platforms and consulting

Gürçe Işıkyıldız
(gurce)
Tools and enhancements

Daniel England
(Mew Pokémon)
Additional code and tools

Hernán Di Pietro
(indiocolifa)
Additional emulation and tools

Roman Standzikowski
(FeralChild)
Open ROMs

Anton Schneider-Michallek
(adtbm)
Presentation and support

Reporting Errors and Omissions

This book is being continuously refined and improved upon by the MEGA65 community. The version of this edition is:

```
commit 06a35d2851767b1d9a560c8c33b3b7a1770f5e4a
date: Sun May 5 15:33:25 2024 +0930
```

We want this book to be the best that it possibly can. So if you see any errors, find anything that is missing, or would like more information, please report them using the MEGA65 User's Guide issue tracker:

https://github.com/mega65/mega65-user-guide/issues

You can also check there to see if anyone else has reported a similar problem, while you wait for this book to be updated.

Finally, you can always download the latest versions of our suite of books from these locations:

- https://mega65.org/mega65-book
- https://mega65.org/user-guide
- https://mega65.org/developer-guide
- https://mega65.org/basic65-ref
- https://mega65.org/chipset-ref
- https://mega65.org/docs

MEGA65 CHIPSET REFERENCE

Published by
the MEGA Museum of Electronic Games & Art e.V., Germany.

WORK IN PROGRESS

May 5, 2024

Contents

CHAPTER 1

System Memory Map

- Introduction
- MEGA65 Native Memory Map
- $D000 – $DFFF I/O Personalities
- CPU Memory Banking
- C64/C65 ROM Emulation

INTRODUCTION

The MEGA65 computer has a large 28-bit address space, which allows it to address up to 256MB of memory and memory-mapped devices. This memory map has several different views, depending on which mode the computer is operating in. Broadly, there are five main modes: (1) Hypervisor mode; (2) C64 compatibility mode; (3) C65 compatibility mode; (4) UltiMAX compatibility mode; and (5) MEGA65-mode, or one of the other modes, where the programmer has made use of MEGA65 enhanced features.

It is important to understand that, unlike the C128, the C65 and MEGA65 allow access to all enhanced features from C64-mode, if the programmer wishes to do so. This means that while we frequently talk about "C64-mode," "C65-mode" and "MEGA65-mode," these are simply terms of convenience for the MEGA65 with its memory map (and sometimes other features) configured to provide an environment that matches the appropriate mode. The heart of this is the MEGA65's flexible memory map.

In this appendix, we will begin by describing the MEGA65's native memory map, that is, where all of the memory, I/O devices and other features appear in the 28-bit address space. We will then explain how C64 and C65 compatible memory maps are accessed from this 28-bit address space.

MEGA65 NATIVE MEMORY MAP

The First Sixteen 64KB Banks

The MEGA65 uses a similar memory map to that of the C65 for the first MB of memory, i.e., 16 memory banks of 64KB each. This is because the C65's 4510 CPU can access only 1MB of address space. These banks can be accessed from BASIC 65 using the **BANK**, **DMA**, **PEEK** and **POKE** commands. The following table summarises the contents of the first 16 banks:

HEX	DEC	Address	Contents
0	0	$0xxxx	First 64KB RAM. This is the RAM visible in C64-mode.
1	1	$1xxxx	Second 64KB RAM. This is the 2nd 64KB of RAM present on a C65.
2	2	$2xxxx	First half of C65 ROM (C64-mode and shared components) *or* RAM
3	3	$3xxxx	Second half of C65 ROM (C65-mode components) *or* RAM
4	4	$4xxxx	Additional RAM (384KB or larger chip-RAM models)
5	5	$5xxxx	Additional RAM (384KB or larger chip-RAM models)
6	6	$6xxxx	Additional RAM (*512KB or larger chip-RAM models)
7	7	$7xxxx	Additional RAM (*512KB or larger chip-RAM models)
8	8	$8xxxx	Additional RAM (*1MB or larger chip-RAM models)
9	9	$9xxxx	Additional RAM (*1MB or larger chip-RAM models)
A	10	$Axxxx	Additional RAM (*1MB or larger chip-RAM models)
B	11	$Bxxxx	Additional RAM (*1MB or larger chip-RAM models)
C	12	$Cxxxx	Additional RAM (*1MB or larger chip-RAM models)
D	13	$Dxxxx	Additional RAM (*1MB or larger chip-RAM models)
E	14	$Exxxx	Additional RAM (*1MB or larger chip-RAM models)
F	15	$Fxxxx	Additional RAM (*1MB or larger chip-RAM models)

* Note that the MEGA65 presently only provides a model featuring 384KB of chip-RAM. Future models may feature larger amounts of chip-RAM (such as 512KB and 1MB).

The key features of this address space are the 128KB of RAM in the first two banks, which is also present on the C65. If you intend to write programs which can also run on a C65, you should only use these two banks of RAM.

On all models it is possible to use all or part of the 128KB of "ROM" space as RAM. To do this, you must first request that the Hypervisor removes the read-only protection on this area, before you will be able to change its contents. If you are writing a program which will start from C64-mode, or otherwise switch to using the C64 part of the ROM, instead of the C65 part), then the second half of that space, i.e., BANK 3, can be safely used for your programs. This gives a total of 192KB of RAM, which is available on all models of the MEGA65.

On models that have 384KB or more of chip RAM, BANK 4 and 5 are also available. Similarly, models which provide 1MB or more of chip RAM will have BANK 6 through 15 also available, giving a total of 896KB (or 960KB, if only the C64 part of the ROM is required) of RAM available for your programs. Note that the MEGA65's built-in freeze cartridge currently freezes only the first 384KB of RAM.

Colour RAM

The MEGA65's VIC-IV video controller supports much larger screens than the VIC-II or VIC-III. For this reason, it has access to a separate colour RAM, similar to on the C64. For compatibility with the C65, the first two kilo-bytes of this are accessible at $1F800 - $1FFFF. The full 32KB or 64KB of colour RAM is located at $FF80000. This is most easily accessed through the use of advanced DMA operations, or the 32-bit base-page indirect addressing mode of the processor.

At the time of writing, the **BANK** and **DMA** commands cannot be used to access the rest of the colour RAM, because the colour RAM is not located in the first mega-byte of address space. This may be corrected in a future revision of the MEGA65, allowing access to the full colour RAM via BANK 15 or an equivalent DMA job.

Additional RAM

Apart from the 384kb of chip-RAM found as standard on all MEGA65 models, most models (devkit, release boards and xemu, but NOT on Nexys boards currently) also have an extra 8MB of RAM starting at $8000000, referred to as 'ATTIC RAM'. It is not visible to the other chips (vic/sid/etc) and can't be used for audio DMA, but code can run from it (more slowly) or it can be used to store content and DMA it in/out of the chip-RAM.

There are also plans underway to support a PMOD hyperRAM module (installed via the trapdoor beneath the MEGA65) in order to provide a further 8MB of RAM starting at $8800000, referred to as 'CELLAR RAM'.

28-bit Address Space

In addition to the C65-style 1MB address space, the MEGA65 extends this to 256MB, by using 28-bit addresses. The following shows the high-level layout of this address space.

HEX	DEC	Size	Contents
0000000	0	1	CPU I/O Port Data Direction Register
0000001	1	1	CPU I/O Port Data
0000002 - 005FFFF	2 - 384KB	384KB	Fast chip RAM (40MHz)
0060000 - 0FFFFFF	384KB - 16MB	15.6MB	Reserved for future chip RAM expansion
1000000 - 3FFFFFF	16MB - 64MB	48MB	Reserved
4000000 - 7FFFFFF	64MB - 128MB	64MB	Cartridge port and other devices on the slow bus (1 - 10 MHz)
8000000 - 87FFFFF	128MB - 135MB	8MB	8MB ATTIC RAM (all models apart from Nexys, presently)
8800000 - 8FFFFFF	135MB - 144MB	8MB	8MB CELLAR RAM (planned PMOD module installed via trapdoor)
9000000 - EFFFFFF	144MB - 240MB	96MB	Reserved for future expansion RAM
F000000 - FF7DFFF	240MB - 255.49MB	15.49MB	Reserved for future I/O expansion
FF7E000 - FF7EFFF	255.49MB - 255.49MB	4KB	VIC-IV Character ROM (write only)
FF80000 - FF87FFF	255.5MB - 255.53MB	32KB	VIC-IV Colour RAM (32KB colour RAM - available on all models)
FF88000 - FF8FFFF	255.53MB - 255.57MB	32KB	Additional VIC-IV Colour RAM (64KB colour RAM - planned to be available on R3 models and beyond)
FF90000 - FFCAFFF	255.53MB - 255.80MB	216KB	Reserved
FFCB000 - FFCBFFF	255.80MB - 255.80MB	4KB	Emulated C1541 RAM
FFCC000 - FFCFFFF	255.80MB - 255.81MB	16KB	Emulated C1541 ROM
FFD0000 - FFD0FFF	255.81MB - 255.81MB	4KB	C64 $Dxxx I/O Personality
FFD1000 - FFD1FFF	255.81MB - 255.82MB	4KB	C65 $Dxxx I/O Personality
FFD2000 - FFD2FFF	255.82MB - 255.82MB	4KB	MEGA65 $Dxxx Ethernet I/O Personality

continued ...

...continued

HEX	DEC	Size	Contents
FFD3000 - FFD3FFF	255.82MB - 255.82MB	4KB	MEGA65 $Dxxx Normal I/O Personality
FFD4000 - FFD5FFF	255.82MB - 255.83MB	8KB	Reserved
FFD6000 - FFD67FF	255.83MB - 255.83MB	2KB	Hypervisor scratch space
FFD6000 - FFD6BFF	255.83MB - 255.83MB	3KB	Hypervisor scratch space
FFD6C00 - FFD6DFF	255.83MB - 255.83MB	512	F011 floppy controller sector buffer
FFD6E00 - FFD6FFF	255.83MB - 255.83MB	512	SD Card controller sector buffer
FFD7000 - FFD70FF	255.83MB - 255.83MB	256	MEGAphone r1 I2C peripherals
FFD7100 - FFD71FF	255.83MB - 255.83MB	256	MEGA65 r2 I2C peripherals
FFD7200 - FFD72FF	255.83MB - 255.83MB	256	MEGA65 HDMI I2C registers (only for R2 and older models fitted with the ADV7511 HDMI driver chip)
FFD7300 - FFD7FFF	255.83MB - 255.84MB	3.25KB	Reserved for future I2C peripherals
FFD8000 - FFDBFFF	255.83MB - 255.86MB	16KB	Hypervisor ROM (only visible in Hypervisor Mode)
FFDC000 - FFDDFFF	255.86MB - 255.87MB	8KB	Reserved for Hypervisor Mode ROM expansion
FFDE000 - FFDE7FF	255.87MB - 255.87MB	2KB	Reserved for Ethernet buffer expansion
FFDE800 - FFDEFFF	255.87MB - 255.87MB	2KB	Ethernet frame read buffer (read only) and Ethernet frame write buffer (write only)
FFDF000 - FFDFFFF	255.87MB - 255.87MB	4KB	Virtual FPGA registers (selected models only)
FFE0000 - FFFFFFF	255.87MB - 256MB	128KB	Reserved

$D000 – $DFFF I/O PERSONALITIES

The MEGA65 supports four different I/O personalities. These are selected by writing the appropriate values to the $D02F KEY register, which is visible in all four I/O personalities. There is more information in *the MEGA65 Book*, C64, C65 and MEGA65 Modes (chapter 11) about the use of the KEY register.

The following table shows which I/O devices are visible in each of these I/O modes, as well as the KEY register values that are used to select the I/O personality.

HEX	C64	C65	MEGA65 ETHERNET	MEGA65
KEY	$00	$A5, $96	$45, $54	$47, $53
$D000 - $D02F	VIC-II	VIC-II	VIC-II	VIC-II
$D030 - $D07F	VIC-II[1]	VIC-III	VIC-III	VIC-III
$D080 - $D08F	VIC-II	F011	F011	F011
$D090 - $D09F	VIC-II	-	SD card	SD card
$D0A0 - $D0FF	VIC-II	RAM EXPAND CONTROL	-	-
$D100 - $D1FF	VIC-II	RED Palette	RED Palette	RED Palette
$D200 - $D2FF	VIC-II	GREEN Palette	GREEN Palette	GREEN Palette
$D300 - $D3FF	VIC-II	BLUE Palette	BLUE Palette	BLUE Palette
$D400 - $D41F	SID Right #1	SID Right #1	SID Right #1	SID Right #1
$D420 - $D43F	SID Right #2	SID Right #2	SID Right #2	SID Right #2
$D440 - $D45F	SID Left #1	SID Left #1	SID Left #1	SID Left #1
$D460 - $D47F	SID Left #2	SID Left #2	SID Left #2	SID Left #2
$D480 - $D49F	SID Right #1	SID Right #1	SID Right #1	SID Right #1
$D4A0 - $D4BF	SID Right #2	SID Right #2	SID Right #2	SID Right #2
$D4C0 - $D4DF	SID Left #1	SID Left #1	SID Left #1	SID Left #1
$D4E0 - $D4FF	SID Left #2	SID Left #2	SID Left #2	SID Left #2
$D500 - $D5FF	SID images	-	Reserved	Reserved
$D600 - $D63F	-	UART	UART	UART
$D640 - $D67F	-	UART images	HyperTrap Registers	HyperTrap Registers
$D680 - $D6FF	-	-	MEGA65 Devices	MEGA65 Devices
$D700 - $D7FF	-	-	MEGA65 Devices	MEGA65 Devices
$D800 - $DBFF	COLOUR RAM	COLOUR RAM	ETHERNET Buffer	COLOUR RAM
$DC00 - $DDFF	CIAs	CIAs / COLOUR RAM	ETHERNET Buffer	CIAs / COLOUR RAM
$DE00 - $DFFF	CART I/O	CART I/O	ETHERNET Buffer	CART I/O / SD SECTOR

[1] In the C64 I/O personality, $D030 behaves as on C128, allowing toggling between 1MHz and 2MHz CPU speed.
[2] The additional MEGA65 SIDs are visible in all I/O personalities.
[3] Some models may replace the repeated images of the first four SIDs with four additional SIDs, for a total of 8 SIDs.

CPU MEMORY BANKING

The 45GS02 processor, like the 6502, can only "see" 64KB of memory at a time. Access to additional memory is via a selection of bank-switching mechanisms. For backward-compatibility with the C64 and C65, the memory banking mechanisms for both of these computers are supported on the MEGA65:

1. C65-style MAP instruction banking
2. C65-style $D030 banking
3. C64-style cartridge banking
4. C64-style $00 / $01 banking

The MAP register overrides all other banking mechanisms. This mechanism selects which of the eight 8KB regions of the 16-bit address space $0000 - $FFFF are mapped to other addresses via an offset. If a region is mapped, then the other banking mechanisms do not apply. This is true even if the offset is 0, allowing the 16-bit addresses to access RAM in bank 0 (such as address 0.D000).

C65-style $D030 banking and C64-style $00 / $01 banking both select regions to map to bank 2, which (by default) contains C64 ROM code. These two mechanisms overlap in which regions they can map to ROM. If either mechanism maps a region to ROM (and it is not mapped elsewhere by the MAP register), then it is mapped to ROM.

The following diagram shows the different types of banking that can apply to the different areas of the 64KB that the CPU can see.

MAP	MAP LO (4 x 8KB slabs)	MAP HI (4 x 8KB slabs)				
I/O/CART		CART ROMLO	CART ROMHI		I/O	CART ROMHI
D030		CHARROM	BASIC	INTER-FACE		KERNAL
C64			BASIC		CHAR ROM	KERNAL
RAM	RAM	RAM	RAM	RAM	RAM	RAM
	$0000 - $7FFF	$8000 - $9FFF	$A000 - $BFFF	$C000 - $CFFF	$D000 - $DFFF	$E000 - $FFFF

There are actually a few further complications. For example, if the cartridge selects the UltiMAX™ game mode, then only the first 4KB of RAM will be visible, and the remaining address space will be un-mapped, and able to be supplied by the cartridge.

C64/C65 ROM EMULATION

The C64 and C65 use ROM memories to hold the KERNAL and BASIC system. The MEGA65 is different: It uses 128KB of its 384KB fast chip RAM at $20000 - $3FFFF

(banks 2 and 3) to hold these system programs. This makes it possible to change or upgrade the "ROM" that the MEGA65 is running, without having to open the computer. It is even possible to use the MEGA65's Freeze Menu to change the "ROM" being used while a program is running.

The C64 and C65 memory banking methods use this 128KB of area when making ROM banks visible. When the RAM banks are mapped, they are always read-only. However, if the MAP instruction or DMA is used to access that address area, it is possible to write to it. For improved backward compatibility, the whole 128KB region of memory is normally set to read-only.

A program can, however, request read-write access to this 128KB area of memory, so that it can make full use of the MEGA65's 384KB of chip RAM. This is accomplished by triggering the *Toggle Rom Write-protect* system trap of the hypervisor. The following code-fragment demonstrates how to do this. Calling it a second time will re-activate the write-protection.

```
LDA #$70
STA $D640
NOP
```

This fragment works by calling sub-function $70 (toggle ROM write-protect) of Hypervisor trap $00. Note that the `NOP` is mandatory. The MEGA65 I/O personality must be first selected, so that the $D640 register is un-hidden.

The current write-protection state can be tested by attempting to write to this area of memory. Also, you can examine and toggle the current state in the MEGA65 Freeze Menu.

NOTE: If you are starting your program from C65-mode, you must first make sure that the I/O area is visible at $D000-$DFFF. The simplest way to do this is to use the `MAP` instruction with all zero values in the registers. The following fragment demonstrates this, and also makes sure that the MEGA65 I/O context is active, so that the hypervisor trap will be able to trigger:

```
; Clear C65 memory map
LDA #$00
TAX
TAY
TAZ
MAP
; Bank I/O in via C64 mechanism
LDA #$35
STA $01
; Do MEGA65 / VIC-IV I/O knock
LDA #$47
STA $D02F
LDA #$53
STA $D02F
; End MAP sequence, thus allowing interrupts to occur again
EOM
; Do Hypervisor call to un-write-protect the ROM area
LDA #$70
STA $D640
NOP
```

C65 Compatibility ROM Layout

The layout of the C65 compatibility 128KB ROM area is identical to that of the C65:

HEX	Contents
$3E000 -- $3FFFF	C65 KERNAL
$3D000 -- $3DFFF	CHARSET B
$3C000 -- $3CFFF	RESERVED
$38000 -- $3BFFF	C65 BASIC GRAPHICS ROUTINES
$32000 -- $37FFF	C65 BASIC
$30000 -- $31FFF	MONITOR (gets mapped at $6000 -- $7FFF)
$2E000 -- $2FFFF	C64 KERNAL
$2D000 -- $2DFFF	CHARSET C
$2C000 -- $2CFFF	INTERFACE
$2A000 -- $2BFFF	C64 BASIC
$29000 -- $29FFF	CHARSET A
$24000 -- $28FFF	RESERVED
$20000 -- $23FFF	DOS (gets mapped at $8000 -- $BFFF)

The INTERFACE program is a series of routines that are used by the C65 to switch between C64-mode, C65-mode and the C65's built-in DOS. The DOS is located in the lower-eighth of the ROM.

CHAPTER 2

VIC-IV Video Interface Controller

- Features
- VIC-II/III/IV Register Access Control
- Video Output Formats, Timing and Compatibility
- Memory Interface
- Hot Registers
- New Modes
- Sprites
- VIC-III Errata Level

- VIC-II / C64 Registers
- VIC-III / C65 Registers
- VIC-IV / MEGA65 Specific Registers

FEATURES

The VIC-IV is a fourth generation Video Interface Controller developed especially for the MEGA65, and featuring very good backwards compatibility with the VIC-II that was used in the C64, and the VIC-III that was used in the C65. The VIC-IV can be programmed as though it were either of those predecessor systems. In addition it supports a number of new features. It is easy to mix older VIC-II/III features with the new VIC-IV features, making it easy to transition from the VIC-II or VIC-III to the VIC-IV, just as the VIC-III made it easy to transition from the VIC-II. Some of the new features and enhancements of the VIC-IV include:

- **Direct access to 384KB RAM** (up from 16KB/64KB with the VIC-II and 128KB with the VIC-III).
- Support for **32KB of 8-bit Colour/Attribute RAM** (up from 2KB on the VIC-III), to support very large screens.
- **HDTV 720×576 / 800×600 native resolution** at both 50Hz and 60Hz for **PAL and NTSC**, with **VGA and digital video** output.
- **81MHz pixel clock** (up from ~ 8MHz with the VIC-II/III), which enables a wide range of new features.
- New 16-colour (16×8 pixels per character cell) and 256-colour (8×8 pixels per character cell) **full-colour text modes**.
- Support for up to **8,192 unique characters in a character set**.
- **Four 256-colour palette banks** (versus the VIC-III's single palette bank), each supporting **23-bit colour depth** (versus the VIC-III's 12-bit colour depth), and which can be rapidly alternated to create even more colourful graphics than is possible with the VIC-III.
- Screen, bitmap, colour and character data can be positioned at any **address with byte-level granularity** (compared with fixed 1KB - 16KB boundaries with the VIC-II/III).
- **Virtual screen dimensioning**, which combined with byte-level data position granularity provides effective **hardware support for scrolling and panning in both X and Y directions**.
- **New sprite modes**: Bitplane modification, **full-colour** (15 foreground colours + transparency) and tiled modes, allowing a wide variety of new and exciting sprite-based effects.
- The ability to stack sprites in a bit-planar manner to produce **sprites with up to 256 colours**.
- Sprites can use 64 bits of data per raster line, allowing **sprites to be 64 pixels wide** when using VIC-II/III mono/multi-colour mode, or 16 pixels wide when using the new VIC-IV full-colour sprite mode.

- **Sprite tile mode**, which allows a sprite to be repeated horizontally across an entire raster line, allowing sprites to be used to create animated backgrounds in a memory-efficient manner.
- Sprites can be configured to use a **separate 256-colour palette** to that used to draw other text and graphics, allowing for a more colourful display.
- **Super-extended attribute mode** which uses two screen RAM bytes and two colour RAM bytes per character mode, which supports a wide variety of new features including **alpha-blending/anti-aliasing**, **hardware kerning/variable-width characters**, hardware horizontal/vertical flipping, alternate palette selection and other powerful features that make it easy to create highly dynamic and colourful displays.
- **Raster-Rewrite Buffer** which allows **hardware-generated pseudo-sprites**, similar to "bobs" on Amiga™ computers, but with the advantage that they are rendered in the display pipeline, and thus do not need to be un-drawn and re-drawn to animate them.
- **Multiple 8-bit colour play-fields** are also possible using the Raster-Rewrite Buffer.

In short, the VIC-IV is a powerful evolution of the VIC-II/III, while retaining the character and distinctiveness of the VIC-series of video controllers.

For a full description of the additional registers that the VIC-IV provides, as well as documentation of the legacy VIC-II and VIC-III registers, refer to the corresponding sections of this appendix. The remainder of the appendix will focus on describing the capabilities and use of many of the VIC-IV's new features.

VIC-II/III/IV REGISTER ACCESS CONTROL

Because the new features of the VIC-IV are all extensions to the existing VIC-II/III designs, there is no concept of having to select the mode in which the VIC-IV will operate: It is always in VIC-IV mode. However, for backwards compatibility with software, the many additional registers of the VIC-IV can be hidden, so that it appears to be either a VIC-II or VIC-III. This is done in the same manner that the VIC-III uses to hide its new features from legacy VIC-II software.

The mechanism is the VIC-III write-only KEY register ($D02F, 53295 decimal). The VIC-III by default conceals its new features until a "knock" sequence is performed. This consists of writing two special values one after the other to $D02F. The following table summarises the knock sequences supported by the VIC-IV, and indicates which are VIC-IV specific, and which are supported by the VIC-III:

First Value Hex (Decimal)	Second Value Hex (Decimal)	Effect	VIC-IV Specific?
$00 (0)	$00 (0)	Only VIC-II registers visible (all VIC-III and VIC-IV new registers are hidden)	No
$A5 (165)	$96 (150)	VIC-III new registers visible	No
$47 (71)	$53 (83)	Both VIC-III and VIC-IV new registers visible	Yes
$45 (69)	$54 (84)	No VIC-II/III/IV registers visible. 45E100 Ethernet controller buffers are visible instead	Yes

Detecting VIC-II/III/IV

Detecting which generation of the VIC-II/III/IV a machine is fitted with can be important for programs that support only particular generations, or that wish to vary their graphical display based on the capabilities of the machine. While there are many possibilities for this, the following is a simple and effective method. It relies on the fact that the VIC-III and VIC-IV do not repeat the VIC-II registers throughout the I/O address space. Thus while $D000 and $D100 are synonymous when a VIC-II is present (or a VIC-III/IV is hiding their additional registers), this is not the case when a VIC-III or VIC-IV is making all of its registers visible. Therefore presence of a VIC-III/IV can be determined by testing whether these two locations are aliases for the same register, or represent separate registers. The detection sequence consists of using the KEY register to attempt to make either VIC-IV or VIC-III additional registers visible. If either succeeds, then we can assume that the corresponding generation of VIC is installed. As the VIC-IV supports the VIC-III KEY knocks, we must first test for the presence of a VIC-IV. Also, we assume that the MEGA65 starts in VIC-IV mode, even when running C65 BASIC. Thus the test can be done in BASIC from either C64 or C65-mode as follows:

```
0 REM IN C65-MODE WE CANNOT SAFELY WRITE TO $D02F, SO WE TEST A DIFFERENT WAY
10 IF PEEK($D018) AND 32 THEN GOTO 65
20 POKE $D000,1:POKE $D02F,71:POKE $D02F,83
30 POKE $D000+256,0:IF PEEK($D000)=1 THEN PRINT"VIC-IV PRESENT":END
40 POKE $D000,1:POKE $D02F,165:POKE $D02F,150
50 POKE $D000+256,0:IF PEEK($D000)=1 THEN PRINT"VIC-III PRESENT":END
60 PRINT "VIC-II PRESENT":END
65 REM WE ASSUME WE HAVE A C65 HERE
70 V1=PEEK($D050):V2=PEEK($D050):V3=PEEK($D050)
80 IF V1<>V2 OR V1<>V3 OR V2<>V3 THEN PRINT "VIC-IV PRESENT":END
90 GOTO 40
```

Line 10 of this program checks whether the screen is a multiple of 2KB. As the screen on the C64 is located at 1KB, this test will fail, and execution will continue to line 20. Line 20 writes 1 to one of the VIC-II sprite position registers, 53248, before writing the MEGA65 knock to the key register, 53295. Line 30 writes to 53248 + 256, which on the C64 is a mirror of 53248, but on a MEGA65 with VIC-IV I/O enabled will be one of the red palette registers. After writing to 53248 + 256, the program checks if the register at 53248 has been modified by the write to 53248 + 256. If it has, then the two addresses point to the same register. This will happen on either a C64 or C65, but not on a computer with a VIC-IV. Thus if 53248 has not changed, we report that we have detected a VIC-IV. If writing to 53248 + 256 did change the value in register 53248, then we proceed to line 40, which writes to 53248 again, and this time writes the VIC-III knock to the key register. Line 50 is like line 30, but as it appears after a VIC-III knock, it allows the detection of a VIC-III. Finally, if neither a VIC-IV nor VIC-III is detected, we conclude that only a VIC-II must be present.

As the MEGA65 is the only C64-class computer that is fitted with a VIC-IV, this can be used as a *de facto* test for the presence of a MEGA65 computer. Detection of a VIC-III can be similarity assumed to indicate the presence of a C65.

VIDEO OUTPUT FORMATS, TIMING AND COMPATIBILITY

Integrated Marvellous Digital Hookup™ (IMDH™) Digital Video Output

The MEGA65 features VGA analog video output and Integrated Marvellous Digital Hookup™ (IMDH™). This is different to existing common digital video standards in several key points:

1. We didn't invent a new connector for it: We instead used the most common digital video connector already in use. So your existing cables should work fine!
2. We didn't make it purposely incompatible with any existing digital video standard. So your existing TVs and monitors should work fine!
3. We don't engage in highway-robbery for other vendors to use the IMDH™ digital video standard, by trying to charge them $10,000 every year, just for the permission to be able to sell a single device. This means that the MEGA65 is cheaper for you!
4. The IMDH™ standard does not allow content-protection or other sovereignty eroding flim-flam. If you produced the video, you can do whatever you like with it!

Connecting to Naughty Proprietary Digital Video Standards

There are digital video standards that are completely backwards compared with IMDH™. Fortunately because of IMDH™'s open approach to interoperability, these should, in most cases, function with the MEGA65 without difficulty. Simply find a video cable fits the IMDH™ connector on the back of your MEGA65, and connect it to your MEGA65 and a TV, Monitor or Projector that has the same connector.

However, regrettably, not all manufacturers have submitted their devices for IMDH™ compliance testing with the MEGA65 team. This means that some TVs and Monitors are, unfortunately, not IMDH™ compliant. Thus while most TVs and Monitors will work with the MEGA65, you might find that you need to try a couple to get a satisfactory result. If you do find a monitor that doesn't work with the MEGA65, please let us know, and also report the problem to the Monitor vendor, recommending that they submit their devices for IMDH™ compliance testing.

The VIC-IV was designed for use in the MEGA65 and related systems, including the MEGAphone family of portable devices. The VIC-IV supports both VGA and digital video output, using the non-proprietary IMDH™ interface. It also supports parallel digital video output suitable for driving LCD display panels. Considerable care has been taken to create a common video front-end that supports these three output modes.

For simplicity and accuracy of frame timing for legacy software, the video format is normally based on the HDTV PAL and NTSC 720×576/480 (576p and 480p) modes using a 27MHz output pixel clock. This is ideal for digital video and LCD display panels. However not all VGA displays support these modes, especially 720×576 at 50Hz.

In terms of VIC-II and VIC-III backwards compatibility, this display format has several effects that do not cause problems for most programs, but can cause some differences in behaviour:

1. Because the VIC-IV display is progressive rather than interlaced, two physical raster lines are produced for each logical VIC-II or VIC-III raster line. This means that there are either 63 or 65 cycles per logical double raster, rather than per physical 576p/480p physical raster. This can cause some minor visual artefacts, when programs make assumptions about where on a horizontal line the VIC is drawing when, for example, the border or screen colour is changed.

2. The VIC-IV does not follow the behaviour of the VIC-III, which allowed changes in video modes, e.g., between text and bitmap mode, on characters. Nor does it follow the VIC-II's policy of having such changes take effect immediately. Instead, the VIC-IV applies changes at the start of each raster line. This can cause some minor artefacts.

3. The VIC-IV uses a single-raster rendering buffer which is populated using the VIC-IV's internal 81MHz pixel clock, before being displayed using the 27MHz output pixel clock. This means that a raster lines display content tends to be rendered much earlier in a raster line than on either the VIC-II or VIC-III. This can cause some artefacts with displays, particularly in demos that rely on specific behaviour of the VIC-II at particular cycles in a raster line, for example for effects

such as VSP or FLI. At present, such effects are unlikely to display correctly on the current revision of the VIC-IV. Improved support for these features is planned for a future revision of the VIC-IV.

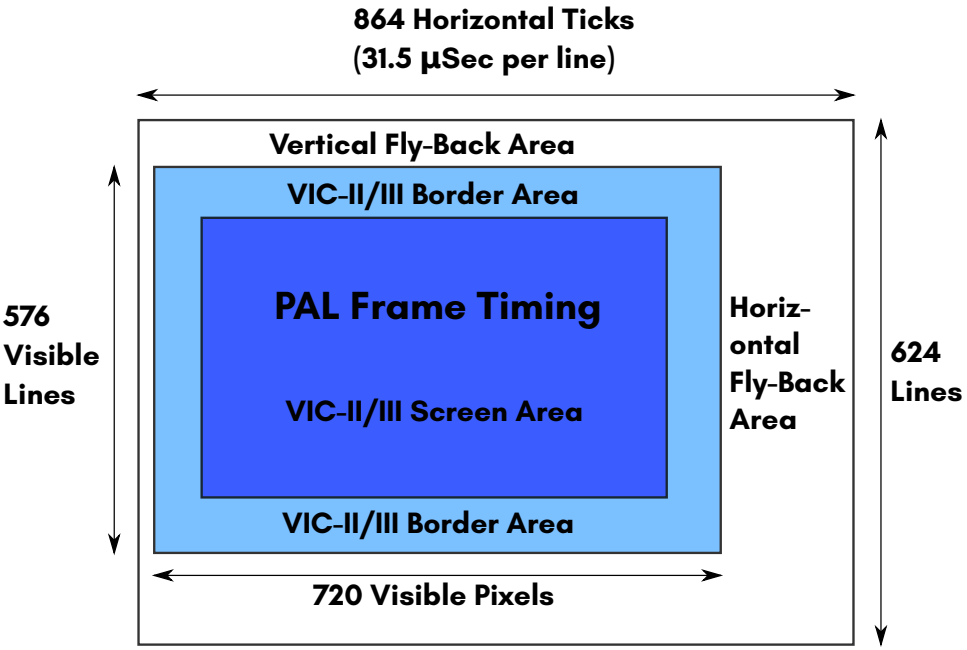
4. The 1280×200 and 1280×400 display modes of the VIC-III are not currently supported, as they cannot be meaningfully displayed on any modern monitor, and no software is known to support or use this feature.

Frame Timing

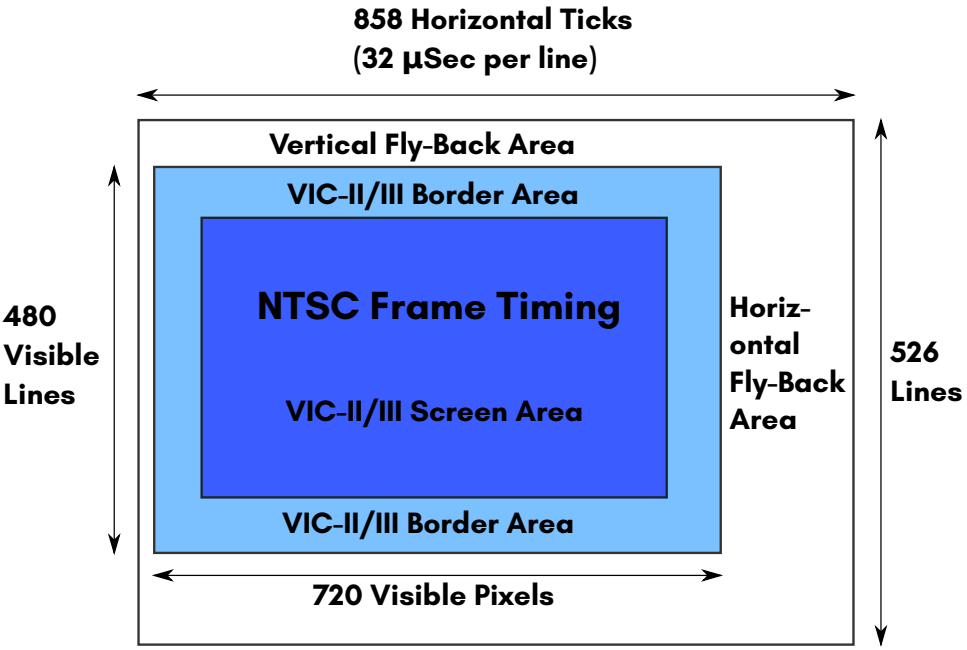
Frame timing is designed to match that of the 6502 + VIC-II combination of the C64. Both PAL and NTSC timing is supported, and the number of cycles per logical raster line, the number of raster lines per frame, and the number of cycles per frame are all adjusted accordingly. To achieve this, the VIC-IV ordinarily uses HDTV 576p 50Hz (PAL) and 480p 60Hz (NTSC) video modes, with timing tweaked to be as close as possible to double-scan PAL and NTSC composite TV modes as used by the VIC-II.

The VIC-IV produces timing impulses at approximately 1MHz which are used by the 45GS02 processor, so that the correct effective frequency is provided when operating at the 1MHz, 2MHz and 3.5MHz C64, C128 and C65 compatibility modes. This allows the single machine to switch between accurate PAL and NTSC CPU timing, as well as video modes. The exact frequency varies between PAL and NTSC modes, to mimic the behaviour of PAL versus NTSC C64, C128 and C65 processor and video timing.

The PAL frame is constructed from 624 physical raster lines, consisting of 864 pixel clock ticks. The pixel clock is 27MHz, which is 1/3 the VIC-IV pixel clock. The visible frame is 720×576 pixels, the entirety of which can be used in VIC-IV mode. In VIC-II and VIC-III modes, the border area reduces the usable size to 640×400 pixels. In VIC-II mode and VIC-III 200H modes, the display is double scanned, with two 31.5 micro-second physical rasters corresponding to a single 63 micro-second VIC-II-style raster line. Thus each frame consists of 312 VIC-II raster lines of 63 micro-seconds each, exactly matching that of a PAL C64.



The NTSC frame is constructed from 526 physical raster lines, consisting of 858 pixel clock ticks. The pixel clock is 27MHz, which is 1/3 the VIC-IV pixel clock. The visible frame is 720×480 pixels, the entirety of which can be used in VIC-IV mode. In VIC-II and VIC-III modes, the border area reduces the usable size to 640×400 pixels. In VIC-II mode and VIC-III 200H modes, the display is double scanned, with two 32 micro-second physical rasters corresponding to a single 64 micro-second VIC-II-style raster line. Thus each frame consists of 263 VIC-II raster lines of 64 micro-seconds each, matching the most common C64 NTSC video timing.



As these HDTV video modes are not supported by all VGA monitors, a compatibility mode is included that provides a 640×480 VGA-style mode. However, as the pixel clock of the MEGA65 is fixed at 27MHz, this mode runs at 63Hz. Nonetheless, this should work on the vast majority of VGA monitors. There should be no problem with the PAL / NTSC modes when using the digital video output of the MEGA65 with the vast majority of IMDH™-enabled monitors and TVs.

To determine whether the MEGA65 is operating in PAL or NTSC, you can enter the Freeze Menu, which displays the current video mode, or from a program you can check the PALNTSC signal (bit 7 of $D06F, 53359 decimal). If this bit is set, then the machine is operating in NTSC mode, and clear if operating in PAL mode. This bit can be modified to change between the modes, e.g.:

```
10 REM ENABLE C65+MEGA65 I/O
20 IF PEEK($D018)<32 THEN POKE $D02F,ASC("G"):POKE $D02F,ASC("S")
30 REM CHECK NTSC BIT
40 NTSC=PEEK($D06F) AND 128
50 REM DISPLAY STATE AND ASK FOR TOGGLE
60 PRINT"MEGA65 IS IN ";:IF NTSC THEN PRINT"NTSC MODE":ELSE PRINT"PAL MODE"
70 INPUT"SWITCH MODES (Y/N)? ",A$
80 REM TOGGLE NTSC BIT
90 IF A$="Y" THEN POKE $D06F,PEEK($D06F) XOR 128:ELSE END
100 REM DISPLAY NEW STATE
110 NTSC=PEEK($D06F) AND 128
120 PRINT"MEGA65 IS IN ";:IF NTSC THEN PRINT"NTSC MODE":ELSE PRINT"PAL MODE"
```

Physical and Logical Rasters

Physical rasters per frame refers to the number of actual raster lines in the PAL or NTSC Enhanced Definition TV (EDTV) video modes used by the MEGA65. Logical Rasters refers to the number of VIC-II-style rasters per frame. Each logical raster consists of two physical rasters per line, since EDTV modes are double-scan modes compared with the original PAL and NTSC Standard Definition TV modes used by the C64. The frame parameters of the VIC-IV for PAL and NTSC are as follows:

Standard	Cycles per Raster	Physical Rasters per Frame	Logical Rasters per Frame
PAL	63	626	312
NTSC	65	526	263

The result is that the frames on the VIC-IV consist of exactly the same number of ~1MHz CPU cycles as on the VIC-II.

Bad Lines

The VIC-IV does not natively incur any "bad lines", because the VIC-IV has its own dedicated memory busses to the main memory and colour RAM of the MEGA65. This means that both the processor and VIC-IV can access the memory at the same time, unlike on the C64 or C65, where they are alternated.

However, to improve compatibility, the VIC-IV signals when a "bad line" would have occurred on the VIC-II. The 45GS02 processor of the MEGA65 accepts these bad line signals, and pauses the CPU for 40 clock cycles, except if the processor is running at full speed, in which case they are ignored. This improves the timing compatibility with the VIC-II considerably. However, the timing is not exact, because the current revision of the 45GS02 pauses for exactly 40 cycles, instead of 40 - 43 cycles, depending on the instruction being executed at the time. Also, the VIC-IV and 45GS02 do not currently pause for sprite fetches.

The bad line emulation is controlled by bit 0 of $D710: setting this bit enables bad line emulation, and clearing it prevents any bad line from stealing time from the processor.

MEMORY INTERFACE

The VIC-IV supports up to 16MB of direct access RAM for video data, however at present, all existing models provide only 384KB of addressable RAM. In MEGA65 systems, the second block of 128KB of RAM (spanning from 128KB-256KB in the memory map) is typically used to hold a C65-compatible ROM, leaving 256KB of RAM available to the user. If software is written to avoid the need to use C65 ROM routines, then the entire 384KB of RAM can be used by the program.

All MEGA65 models presently support 32KB of colour RAM, however there are plans for the latest R3 board to support 64KB of colour RAM (or possibly even 128KB).

The VIC-IV supports all legacy VIC-II and VIC-III methods for accessing this RAM, including the VIC-II's use of 16KB banks, and the VIC-III's Display Address Translator (DAT). This additional memory can be used for character and bitmap displays, as well as for sprites. However, the VIC-III bitplane modes remain limited to using only the first 128KB of RAM, as the VIC-IV does not enhance the bitplane mode.

Startup Base Addresses

If no mappings are changed and no screen memory is relocated (see the following paragraph for more on this) the base addresses can be used to place characters on the screen and set colour and attributes accordingly:

- **$0800 Screen RAM**
- **$ff80000 Colour RAM**

These values are the upper left character on the screen no matter if in 40 or 80 column mode. In BASIC you can use the dollar sign directly to use the hexadecimal format. For a far better method in BASIC see *the MEGA65 Book*, Screen Text and Colour Arrays (subsection B) .

Relocating Screen Memory

To use the additional memory for screen RAM, the screen RAM start address can be adjusted to any location in memory with byte-level granularity by setting the SCRNPTR registers ($D060 - $D063, 53344 - 53347 decimal). For example, to set the screen memory to address 12345:

```
REM ENABLE C65+MEGA65 I/O
IF PEEK($D018)<32 THEN POKE $D02F,ASC("G"):POKE $D02F,ASC("S")
POKE $D060,$45:POKE $D061,$23:POKE $D062,$1
```

Relocating Character Generator Data

The location of the character generator data can also be set with byte-level precision via the CHARPTR registers at $D068 – $D06A (53352 – 53354 decimal). As usual, the first of these registers holds the lowest-order byte, and the last the highest-order byte. The three bytes allow for placement of character data anywhere in the first 16MB of RAM. For systems with less than 16MB of RAM accessible by the VIC-IV, the upper address bits should be zero.

For example, to indicate that character generator data should be sourced beginning at $41200 (266752 decimal), the following could be used. Note that the command WPOKE can be used to write two bytes as a word into a memory or I/O location. Therefore, we use WPOKE to write $00 into $D068 and $12 into $D069, and an additional POKE to write the high byte $A into $D06A by dividing the address by 65536:

```
REM ENABLE C65+MEGA65 I/O
IF PEEK($D018)<32 THEN POKE $D02F,ASC("G"):POKE $D02F,ASC("$")
REM HEX $41200 IS EASILY DIVIDED IN ITS 3 BYTES $00, $12, $4
REM WPOKE SETS THE LOWER TWO BYTES IN ONE COMMAND AND
REM THE FOLLOWING POKE SETS THE UPPER BYTE
A=$41200
WPOKE $D068,A
POKE $D06A,A/65536
```

Relocating Colour / Attribute RAM

The area of colour RAM being used can be similarly set using the COLPTR registers ($D064 – $D065, 53348 – 53349 decimal). That is, the value is an offset from the start of the colour / attribute RAM. This is because, like on the C64, the colour / attribute RAM of the MEGA65 is a separate memory component, with its own dedicated connection to the VIC-IV. By default, the COLPTRs are set to zero, which replicates the behaviour of the VIC-II/III. To set the display to use the colour / attribute RAM beginning at offset $4000, one could use something like:

```
REM ENABLE C65+MEGA65 I/O
IF PEEK($D018)<32 THEN POKE $D02F,ASC("G"):POKE $D02F,ASC("$")
REM SET COLPTR TO $4000, SPLITS INTO $00 LSB and $40 MSB
POKE $D064,$00
POKE $D065,$40
```

Relocating Sprite Pointers and Images

The location of the sprite pointers can also be moved, and sprites can be made to have their data anywhere in first 4MB of memory. This is accomplished by first setting the location of the sprite pointers by setting the SPRPTRADR registers ($D06C – $D06E,

53356 - 53358 decimal, but note that only the bottom 7 bits of $D06E are used, as the highest bit is used for the SPRPTR16 signal). This allows the list of eight sprite pointers to be moved from the end of screen RAM to an arbitrary location in the first 8MB of RAM. SPRPTRADR must be aligned to a 16-byte boundary (a multiple of 16).

To allow sprites themselves to be located anywhere in the first 4MB of RAM, the SPRPTR16 bit in $D06E must be set. In this mode, two bytes are used to indicate the location of each sprite, instead of one. That is, the list of sprite pointers will be 16 bytes long, instead of 8 bytes long as on the VIC-II/III. When SPRPTR16 is enabled, the location of the sprite pointers should always be set explicitly via the SPRPTRADR registers.

For example, to position the sprite pointers at location 800 - 815, you could use something like the following code. Note that a little gymnastics is required to keep the SPRPTR16 bit unchanged, and also to work around the AND binary operator not working with values greater than 65535:

```
REM ENABLE C65+MEGA65 I/O
IF PEEK($D018)<32 THEN POKE $D02F,ASC("G"):POKE $D02F,ASC("S")
POKE $D06C,(800-INT(800/65536)*65536) AND 255
POKE $D06D,INT(800/256) AND 255
POKE $D06E,(PEEK($D06E) AND 128)+INT(800/65536)
```

The location of each sprite image remains a multiple of 64 bytes, thus allowing for up to 65,536 unique sprite images to be used at any point in time, if the system is equipped with sufficient RAM (4MB or more). In this mode, the VIC-II 16KB banking is ignored, and the location of sprite data is simply 64 × the pointer value. For example, to have the data for a sprite at $C000 (49152 decimal), this would be sprite location 768, because 49152 divided by 64 = 768. We then need to split 768 into high and low bytes, to set the two pointer bytes: 768 = 256×3, with remainder 0, so this would require the two sprite pointer bytes to be 0 (low byte, which comes first) and 3 (high byte). Thus if the sprite pointers were located at $7F8 (2040 decimal), setting the first sprite to sprite image 768 could be done with something like:

```
POKE 2040,768-256*INT(768/256)
POKE 2041,INT(768/256)
```

HOT REGISTERS

Some VIC-IV registers support features similar to the VIC-II and VIC-III, but with expanded capabilities. For backwards compatibility, writing to specific VIC-II and VIC-III registers also causes related VIC-IV registers to reset with consistent values. This behavior can be configured with the HOTREG flag in bit 7 of $D05D.

For example, the lower four bits of register $D018 (**CB**) set the VIC-II character set address, as a multiple of 1 KiB. VIC-IV can locate the character set to any 24-bit address

using $D06A (**CHARPTRBNK**), $D068 (**CHARPTRLSB**), and $D069 (**CHARPTRMSB**). If you set CB, the VIC-IV registers will also be updated to match.

The complete set of VIC-II "hot" registers that affect VIC-IV registers include:

- $D011 (53265): RB8, ECM, BMM, BLNK, RSEL, YSCL
- $D016 (53270): RST, MCM, CSEL, XSCL
- $D018 (53272): VS, CB
- $D031 (53297): VIC-III modes: H640, FAST, ATTR, BPM, V400, H1280, MONO, INT
- The VIC-II bank bits of $DD00 (56576) (CIA 2 PORTA)

Whenever any of those registers are modified, even by writing the existing value back into them, all of the corresponding VIC-IV registers will be updated based on the VIC-II register values, even those unrelated to the register that was written to. This includes the FAST flag, which resides in a hot register byte location ($D031) but is unrelated to video parameters and does not have a corresponding VIC-IV register.

The registers that are modified during this process are listed below. Note that some of these registers are internal to the VIC-IV, and cannot be directly queried or modified by the user. Where this is the case, no addresses are listed for the registers.

- **X position of the left side border edge.** This internal register is updated set the left side border to the width indicated in the Single Side Border Width registers ($D05C contains the LSB, and bits 0 – 5 of $D05D contain the MSB of the side border width. Note that the width of the side border is based on the low-level video frame dimensions, not the display screen size of the video mode. The 38/40 column field of $D016 is set to 38 columns, the left border edge will appear 14 pixels to the right of its normal position.
- **X position of the right side border edge.** This is the same as the left side border edge, but for the right-hand edge of the screen. Note that if the 38/40 column flag is set to 38 columns, that the right border edge is moved 17 pixels to the left of its normal position.
- **Y position of the top border edge ($D048 LSB, bits 0 – 3 of $D049 MSB).** This internal register is set to the normal top position of the screen, minus the value of the RASLINE0 field in bits 0 – 5 of $D06F. If the 24/25 rows field of $D011 is set to 24 rows, then the edge of the top border will be lowed by 8 raster lines.
- **Y position of the bottom border edge ($D04A LSB, bits 0 – 3 of $D04B MSB).** This internal register is set to the normal top position of the screen, plus 400 raster lines, to create the normal 400px tall primary display area within the borders. If the 24/25 rows field of $D011 is set to 24 rows, then the edge of the top border will be raised by 8 raster lines.

- **Character Generator Vertical Scale ($D05B).** This register is set to 0 for V200 or 1 for V400 modes, to cause each row of pixels in a character to be either 1 or 2 pixels tall, respectively, according to the V400 flag.
- **Number of character rows to display ($D07B).** This register is set to either 25-1 = 24 or 50-1 = 49 to display either 25 or 50 rows of text. Note that when $D011 is used to bring the vertical borders inwards to reduce the number of visible character rows, that the VIC-IV still draws all 25 or 50 rows.
- **X Position Where Character Display Starts ($D04C LSB, bits 0 - 3 of $D04D MSB).** This register is set to a position relative to the edge of the 40-column wide text display, plus 2× the smooth scrolling position indicated in $D016.
- **Y Position Where Character Display Starts ($D04E LSB, buts 0 - 3 of $D04F MSB).** This register is set to the top edge of the vertical border, minus the VIC-II First Raster adjustment register (bits 0 -5 of $D06F), plus any offset due to the vertical smooth-scroll bits in $D011.
- **Virtual Row Width ($D058 LSB, $D059 MSB), i.e., the number of bytes of screen and colour RAM that the VIC-IV advances when displaying each successive row of characters.** This register is set to 40 if the H640 flag is clear, or to 80 if the H640 flag is set, making the advance match the number of characters to be displayed.
- **Display Row Width ($D05E LSB, bits 4 - 5 of $D063 MSB).** This register is set to 40 if the H640 flag is clear, or to 80 if the H640 flag is set.
- **Base Address of Screen RAM ($D060 - $D062, representing a 24-bit address).** This address is reset to the address as computed by reference to $D018 and $DD00, as on the C64.
- **VIC-II Sprite Pointer Address ($D06C - $D06E, representing a 24-bit address).** This register is reset to the normal location at the end of the screen memory of the current mode. If the H640 flag is set, then this will be at the end of 2KB screen RAM area, or if the H640 flag is not set, it will point to the end of the 1KB screen RAM area, as on a C64.
- **Character Set Base Address ($D068 - $D06A, representing a 24-bit address).** Note that the hot register function sets only the lower 16 bits of the character set address. That is, $D06A is not cleared. This is an intentional behaviour, that makes it easier to replace the character set in existing VIC-II-oriented software with another character set in another bank of RAM.
- **Colour RAM Base Address ($D064 LSB, $D065 MSB).** These registers are reset to zero, causing the VIC-IV to expect the colour RAM for the screen to be in the first part of the colour RAM, to be compatible with the VIC-II and VIC-III.

This behavior of the VIC-II registers is intended primarily for legacy software that is not aware of (and therefore never writes to) VIC-IV registers. Hot register propagation can be disabled by clearing the HOTREG ("hot register") signal: bit 7 of $D05D (53341).

If you clear the HOTREG flag, you will need to update VIC-IV registers directly when you wish to change video mode parameters.

If you make a change to a hot register while HOTREG is disabled then re-enable HOTREG, all hot registers update immediately. There are rare cases where you might want to make a change to a hot register without updating VIC-IV registers but also want hot registers enabled after the change. To do this, you can cancel the pending update by writing a 0 to HOTREG again just before re-enabling hot registers. The full procedure is:

1. Write 0 to HOTREG to disable hot registers.
2. Update the VIC-II register you want to change.
3. Write 0 to HOTREG to cancel the pending update.
4. Write 1 to HOTREG to re-enable hot registers.

In assembly language:

```
lda #%10000000
trb $d05d        ; disable hot registers

lda #%01000000
tsb $d031        ; update a VIC-II/III register

lda #%10000000
trb $d05d        ; clear pending update
tsb $d05d        ; re-enable hot registers
```

NEW MODES

Why the new VIC-IV modes are Character and Bitmap modes, not Bitplane modes

The new VIC-IV video modes are derived from the VIC-II character and bitmap modes, rather than the VIC-III bitplane modes. This decision was based on several realities of programming a memory-constrained 8-bit home computer:

1. Bitplanes require that the same amount of memory is given to each area on screen, regardless of whether it is showing empty space, or complex graphics. There is no way with bitplanes to reuse content from within an image in another part of the image. However, most C64 games use highly repetitive displays, with common elements appearing in various places on the screen, of which Boulder Dash and Super Giana Sisters would be good examples.
2. Bitplanes also make it difficult to update a display, because every pixel is unique, in that there is no way to make a change, for example to the animation in an

onscreen element, and have it take effect in all places at the same time. The diamond animations in Boulder Dash are a good example of this problem. The requirement to modify multiple separate bytes in each bitplane create an increased computational burden, which is why there were calls for the Amiga AAA chip-set to include so-called "chunky" modes, rather than just bitplane based modes. While the Display Address Translator (DAT) and DMAgic of the C65 provide some relief to this problem, the relief is only partial.

3. Scrolling using the C65 bitplanes requires copying the entire bitplane, as the hardware support for smooth scrolling does not extend to changing the bitplane source address in a fine manner. Even using the DMAgic to assist, scrolling a 320×200 256-colour display requires 128,000 clock cycles in the best case (reading and writing 320×200 = 64000 bytes). At 3.5MHz on the C65 this would require about 36 milli-seconds, or about 2 complete video frames. Thus for smooth scrolling of such a display, a double buffered arrangement would be required, which would consume 128,000 of the 131,072 bytes of memory.

   In contrast, the well known character modes of the VIC-II are widely used in games, due to their ability to allow a small amount of screen memory to select which 8×8 block of pixels to display, allowing very rapid scrolling, reduced memory consumption, and effective hardware acceleration of animation of common elements. Thus the focus of improvements in the VIC-IV has been on character mode. As bitmap mode on the VIC-II is effectively a special case of character mode, with implied character numbers, it comes along free for the ride on the VIC-IV, and will only be mentioned in the context of a very few bitmap-mode specific improvements that were trivial to make, and it thus seemed foolish to not implement, in case they find use.

Displaying more than 256 unique characters via "Super-Extended Attribute Mode"

The primary innovation is the addition of the Super-Extended Attribute Mode. The VIC-II already uses 12 bits per character: Each 8×8 cell is defined by 12 bits of data: 8 bits of screen RAM data, by default from $0400 - $07E7 (1024 - 2023 decimal), indicating which characters to show, and 4 bits of colour data from the 1K nibble colour RAM at $D800 - $DBFF (55296 - 56319 decimal). The VIC-III of the C65 uses 16 bits, as the colour RAM is now 8 bits, instead of 4, with the extra 4 bits of colour RAM being used to support attributes (blink, bold, underline and reverse video). It is recommended to revise how this works, before reading the following. A good introduction to the VIC-II text mode can be found in many places. Super-Extended Attribute mode doubles the number of bits per character used from the VIC-III's 16, to 32: Two bytes of screen RAM and two bytes of colour/attribute RAM.

Super-Extended Attribute Mode is enabled by setting bit 0 in $D054 (53332 decimal). Remember to first enable VIC-IV mode, to make this register accessible. When this bit is set, two bytes are used for each of the screen memory and colour RAM for each character shown on the display. Thus, in contrast to the 12 bits of information that the C64 uses per character, and the 16 bits that the VIC-III uses, the VIC-IV has 32

bits of information. How those 32 bits are used varies slightly among the particular modes, as described in the following tables, including whether the GOTOX bit is set.

Note also that enabling BOLD and REVERSE attributes at the same time on the MEGA65 selects an alternate palette, effectively allowing 512 colours on screen, but each 8×8 character can use colours only from one 256 colour palette.

Default Bit Fields (when GOTOX bit is cleared):

Bit(s)	Function when GOTOX bit is cleared
Screen RAM byte 0	
Bits 7 - 0	Low byte of character number, the same as the VIC-II and VIC-III
Screen RAM byte 1	
Bits 7 - 5	Trim pixels from right-hand side of character (bits 0 - 2)
Bits 4 - 0	Upper 5 bits of character number (bits 8 - 12), allowing addressing of 8,192 unique characters
Colour RAM byte 0	
Bit 7	Vertically flip the character
Bit 6	Horizontally flip the character
Bit 5	Enable alpha blend mode, pixel values are treated as alpha values blending between foreground colour and background colour (needs bit 7 of $d054 set)
Bit 4	**GOTOX is cleared (set to 0)** GOTOX allows repositioning of characters along a raster via the Raster-Rewrite Buffer, discussed later). Must be set to 0 for displaying characters - when set, it moves the position where the next character will be drawn, without actually drawing anything. See the following table for more explanation of this mode.
Bit 3	If set, Full-Colour characters use 4 bits per pixel and are 16 pixels wide (less any right-hand side trim bits), instead of using 8 bits per pixel. When using 8 bits per pixels, the characters are the normal 8 pixels wide
Bit 2	Trim pixels from right-hand side of character (bit 3)
Bits 1 - 0	Number of pixels to trim from top or bottom of character
Colour RAM byte 1	
If VIC-II multi-colour mode is enabled:	
Bits 7 - 4	Upper 4 bits of colour of character
If VIC-III extended attributes are enabled:	
Bit 7	Hardware underlining of character
Bit 6	Hardware bold attribute of character *
Bit 5	Hardware reverse video enable of character *
Bit 4	Hardware blink of character
Remaining bit-field is common:	
Bits 3 - 0	Low 4 bits of colour of character

Bit Fields when GOTOX bit is set:

Bit(s)	Function when GOTOX bit is set
Screen RAM byte 0	
Bits 7 - 0	Lower 8 bits of new X position to start drawing the next character, relative to the start of character drawing. Setting to 0 causes the next character to be drawn over the top of the left-most character.
Screen RAM byte 1	
Bits 7 - 5	FCM Character data Y offset: Characters display normally when set to zero. When non-zero, 8 × the value is added to the character address. With careful planning, this can be used to smoothly vertically scroll multiple layers of RRB content.
Bits 4 - 3	RESERVED, set to 0
Bits 1 - 0	Upper 2 bits of new X position (Highest bit is 2's complement signed bit)
Colour RAM byte 0	
Bit 7	If set, then background/transparent pixels will not be drawn for subsequent characters, allowing layering
Bit 6	If set, the following characters will be rendered as background, allowing sprites to appear in front of them, even when sprites are set to background.
Bit 5	RESERVED, set to 0
Bit 4	**GOTOX, set to 1**. GOTOX allows repositioning of characters along a raster via the Raster-Rewrite Buffer, discussed later).
Bit 3	ROWMASK. If set, then the pixel row mask is used to determine which pixel rows of the following characters should be rendered. This can be used to vertically scroll characters using the Raster-Rewrite Buffer, by drawing each character twice, once shifted down on the screen line on which it appears, and a second time, shifted up in the following screen line, and masked so that only the pixel rows belonging to the scrolled character are displayed, and not data from either before or after that character's data.
Bit 2	If set, the following characters will be drawn as foreground, regardless of their colour, allowing sprites to appear behind them.
Bits 1 - 0	RESERVED, set to 0
Colour RAM byte 1	
Bits 7 - 0	Pixel row mask flags

We can see that we still have the C64 style bottom 8 bits of the character number in the first screen byte. The second byte of screen memory gets five extra bits for that, allowing 2^{13} = 8,192 different characters to be used on a single screen. That's more than enough for unique characters covering an 80×50 screen (which is possible to create with the VIC-IV). The remaining bits allow for trimming of the character.

This allows for variable width characters, which can be used to do things that would not normally be possible, such as using text mode for free horizontal placement of characters (or parts thereof). This was originally added to provide hardware support for proportional width fonts.

For the colour RAM, the second byte (byte 1) is the same as the C65, i.e., the lower half providing four bits of foreground colour, as on the C64, plus the optional VIC-III extended attributes. The C65 specifications document describes the behaviour when more than one of these are used together, most of which are logical, but there are a few combinations that behave differently than one might expect. For example, combining bold with blink causes the character to toggle between bold and normal mode. Bold mode itself is implemented by effectively acting as bit 4 of the foreground colour value, causing the colour to be drawn from different palette entries than usual.

However, if you do not need VIC-III extended attributes, you can instead use the upper four bits of the second byte of colour RAM to contain more bits for the colour index, allowing selection from the full range of 256 colour entries. This mode is activated by enabling the VIC-II's multi-colour mode while full-colour mode is active.

The C65 / VIC-III attributes and the use of 256 colour 8-bit values for various VIC-II colour registers is enabled by setting bit 5 of $D031 (53297 decimal). Therefore this is highly recommended when using the VIC-IV mode, as otherwise certain functions will not behave as expected. Note that BOLD+REVERSE together has the meaning of selecting an alternate palette on the MEGA65, which differs from the C65.

Many effects are possible due to Super-Extended Attribute Mode. A few possibilities are explained in the following sub-sections.

Using Super-Extended Attribute Mode

Super-Extended Attribute Mode requires double the screen RAM and colour RAM as the VIC-II/III text modes. This is because two bytes of each are required to define each character, instead of one. The screen RAM can be located anywhere in the 384KB of main memory using registers $D060 - $D062 (53344 - 53346 decimal). The colour RAM can be located anywhere in the 32KB colour RAM. Only the first 1 or 2KB of the colour RAM is visible at $D800 - $DBFF or $D800 - $DFFF (if the *CRAM2K* signal is set in bit 0 of $D030, 53296 decimal). Thus if using a screen larger than 40×25 characters use of the DMA controller or some other means is required to access the full amount of colour RAM. Therefor we will initially discuss using Super-Extended Attribute Mode with a 40x25 character display.

The first step is to enable the Super-Extended Attribute Mode by asserting the *FCLRHI* and *CHR16* signals, by setting bits 2 and 0 of $D054 (53332 decimal). As this is a VIC-IV register, we must first enable the VIC-IV I/O mode. The VIC-IV must also be configured to 40 column mode, by clearing the *H640* signal by clearing bit 7 of $D031 (53297 decimal). This is because each pair of characters will be used to form a single character on screen, with one character requiring two screen RAM bytes, thus 80 screen RAM bytes are required to display 40 characters. Similarly 80 colour RAM bytes are required as well.

To understand this visually, it is helpful to first consider the normal C64 screen memory layout:

$400	$401	$402	$403	$404	$405	$406	$407	$408	$409	$40a	$40b	$40c	$40d	$40e	$40f	$410	$411	$412	$413	$414	$415	$416	$417	$418	$419	$41a	$41b	$41c	$41d	$41e	$41f	$420	$421	$422	$423	$424	$425	$426	$427
$428	$429	$42a	$42b	$42c	$42d	$42e	$42f	$430	$431	$432	$433	$434	$435	$436	$437	$438	$439	$43a	$43b	$43c	$43d	$43e	$43f	$440	$441	$442	$443	$444	$445	$446	$447	$448	$449	$44a	$44b	$44c	$44d	$44e	$44f
$450	$451	$452	$453	$454	$455	$456	$457	$458	$459	$45a	$45b	$45c	$45d	$45e	$45f	$460	$461	$462	$463	$464	$465	$466	$467	$468	$469	$46a	$46b	$46c	$46d	$46e	$46f	$470	$471	$472	$473	$474	$475	$476	$477
$478	$479	$47a	$47b	$47c	$47d	$47e	$47f	$480	$481	$482	$483	$484	$485	$486	$487	$488	$489	$48a	$48b	$48c	$48d	$48e	$48f	$490	$491	$492	$493	$494	$495	$496	$497	$498	$499	$49a	$49b	$49c	$49d	$49e	$49f
$4a0	$4a1	$4a2	$4a3	$4a4	$4a5	$4a6	$4a7	$4a8	$4a9	$4aa	$4ab	$4ac	$4ad	$4ae	$4af	$4b0	$4b1	$4b2	$4b3	$4b4	$4b5	$4b6	$4b7	$4b8	$4b9	$4ba	$4bb	$4bc	$4bd	$4be	$4bf	$4c0	$4c1	$4c2	$4c3	$4c4	$4c5	$4c6	$4c7
$4c8	$4c9	$4ca	$4cb	$4cc	$4cd	$4ce	$4cf	$4d0	$4d1	$4d2	$4d3	$4d4	$4d5	$4d6	$4d7	$4d8	$4d9	$4da	$4db	$4dc	$4dd	$4de	$4df	$4e0	$4e1	$4e2	$4e3	$4e4	$4e5	$4e6	$4e7	$4e8	$4e9	$4ea	$4eb	$4ec	$4ed	$4ee	$4ef
$4f0	$4f1	$4f2	$4f3	$4f4	$4f5	$4f6	$4f7	$4f8	$4f9	$4fa	$4fb	$4fc	$4fd	$4fe	$4ff	$500	$501	$502	$503	$504	$505	$506	$507	$508	$509	$50a	$50b	$50c	$50d	$50e	$50f	$510	$511	$512	$513	$514	$515	$516	$517
$518	$519	$51a	$51b	$51c	$51d	$51e	$51f	$520	$521	$522	$523	$524	$525	$526	$527	$528	$529	$52a	$52b	$52c	$52d	$52e	$52f	$530	$531	$532	$533	$534	$535	$536	$537	$538	$539	$53a	$53b	$53c	$53d	$53e	$53f
$540	$541	$542	$543	$544	$545	$546	$547	$548	$549	$54a	$54b	$54c	$54d	$54e	$54f	$550	$551	$552	$553	$554	$555	$556	$557	$558	$559	$55a	$55b	$55c	$55d	$55e	$55f	$560	$561	$562	$563	$564	$565	$566	$567
$568	$569	$56a	$56b	$56c	$56d	$56e	$56f	$570	$571	$572	$573	$574	$575	$576	$577	$578	$579	$57a	$57b	$57c	$57d	$57e	$57f	$580	$581	$582	$583	$584	$585	$586	$587	$588	$589	$58a	$58b	$58c	$58d	$58e	$58f
$590	$591	$592	$593	$594	$595	$596	$597	$598	$599	$59a	$59b	$59c	$59d	$59e	$59f	$5a0	$5a1	$5a2	$5a3	$5a4	$5a5	$5a6	$5a7	$5a8	$5a9	$5aa	$5ab	$5ac	$5ad	$5ae	$5af	$5b0	$5b1	$5b2	$5b3	$5b4	$5b5	$5b6	$5b7
$5b8	$5b9	$5ba	$5bb	$5bc	$5bd	$5be	$5bf	$5c0	$5c1	$5c2	$5c3	$5c4	$5c5	$5c6	$5c7	$5c8	$5c9	$5ca	$5cb	$5cc	$5cd	$5ce	$5cf	$5d0	$5d1	$5d2	$5d3	$5d4	$5d5	$5d6	$5d7	$5d8	$5d9	$5da	$5db	$5dc	$5dd	$5de	$5df
$5e0	$5e1	$5e2	$5e3	$5e4	$5e5	$5e6	$5e7	$5e8	$5e9	$5ea	$5eb	$5ec	$5ed	$5ee	$5ef	$5f0	$5f1	$5f2	$5f3	$5f4	$5f5	$5f6	$5f7	$5f8	$5f9	$5fa	$5fb	$5fc	$5fd	$5fe	$5ff	$600	$601	$602	$603	$604	$605	$606	$607
$608	$609	$60a	$60b	$60c	$60d	$60e	$60f	$610	$611	$612	$613	$614	$615	$616	$617	$618	$619	$61a	$61b	$61c	$61d	$61e	$61f	$620	$621	$622	$623	$624	$625	$626	$627	$628	$629	$62a	$62b	$62c	$62d	$62e	$62f
$630	$631	$632	$633	$634	$635	$636	$637	$638	$639	$63a	$63b	$63c	$63d	$63e	$63f	$640	$641	$642	$643	$644	$645	$646	$647	$648	$649	$64a	$64b	$64c	$64d	$64e	$64f	$650	$651	$652	$653	$654	$655	$656	$657
$658	$659	$65a	$65b	$65c	$65d	$65e	$65f	$660	$661	$662	$663	$664	$665	$666	$667	$668	$669	$66a	$66b	$66c	$66d	$66e	$66f	$670	$671	$672	$673	$674	$675	$676	$677	$678	$679	$67a	$67b	$67c	$67d	$67e	$67f
$680	$681	$682	$683	$684	$685	$686	$687	$688	$689	$68a	$68b	$68c	$68d	$68e	$68f	$690	$691	$692	$693	$694	$695	$696	$697	$698	$699	$69a	$69b	$69c	$69d	$69e	$69f	$6a0	$6a1	$6a2	$6a3	$6a4	$6a5	$6a6	$6a7
$6a8	$6a9	$6aa	$6ab	$6ac	$6ad	$6ae	$6af	$6b0	$6b1	$6b2	$6b3	$6b4	$6b5	$6b6	$6b7	$6b8	$6b9	$6ba	$6bb	$6bc	$6bd	$6be	$6bf	$6c0	$6c1	$6c2	$6c3	$6c4	$6c5	$6c6	$6c7	$6c8	$6c9	$6ca	$6cb	$6cc	$6cd	$6ce	$6cf
$6d0	$6d1	$6d2	$6d3	$6d4	$6d5	$6d6	$6d7	$6d8	$6d9	$6da	$6db	$6dc	$6dd	$6de	$6df	$6e0	$6e1	$6e2	$6e3	$6e4	$6e5	$6e6	$6e7	$6e8	$6e9	$6ea	$6eb	$6ec	$6ed	$6ee	$6ef	$6f0	$6f1	$6f2	$6f3	$6f4	$6f5	$6f6	$6f7
$6f8	$6f9	$6fa	$6fb	$6fc	$6fd	$6fe	$6ff	$700	$701	$702	$703	$704	$705	$706	$707	$708	$709	$70a	$70b	$70c	$70d	$70e	$70f	$710	$711	$712	$713	$714	$715	$716	$717	$718	$719	$71a	$71b	$71c	$71d	$71e	$71f
$720	$721	$722	$723	$724	$725	$726	$727	$728	$729	$72a	$72b	$72c	$72d	$72e	$72f	$730	$731	$732	$733	$734	$735	$736	$737	$738	$739	$73a	$73b	$73c	$73d	$73e	$73f	$740	$741	$742	$743	$744	$745	$746	$747
$748	$749	$74a	$74b	$74c	$74d	$74e	$74f	$750	$751	$752	$753	$754	$755	$756	$757	$758	$759	$75a	$75b	$75c	$75d	$75e	$75f	$760	$761	$762	$763	$764	$765	$766	$767	$768	$769	$76a	$76b	$76c	$76d	$76e	$76f
$770	$771	$772	$773	$774	$775	$776	$777	$778	$779	$77a	$77b	$77c	$77d	$77e	$77f	$780	$781	$782	$783	$784	$785	$786	$787	$788	$789	$78a	$78b	$78c	$78d	$78e	$78f	$790	$791	$792	$793	$794	$795	$796	$797
$798	$799	$79a	$79b	$79c	$79d	$79e	$79f	$7a0	$7a1	$7a2	$7a3	$7a4	$7a5	$7a6	$7a7	$7a8	$7a9	$7aa	$7ab	$7ac	$7ad	$7ae	$7af	$7b0	$7b1	$7b2	$7b3	$7b4	$7b5	$7b6	$7b7	$7b8	$7b9	$7ba	$7bb	$7bc	$7bd	$7be	$7bf
$7c0	$7c1	$7c2	$7c3	$7c4	$7c5	$7c6	$7c7	$7c8	$7c9	$7ca	$7cb	$7cc	$7cd	$7ce	$7cf	$7d0	$7d1	$7d2	$7d3	$7d4	$7d5	$7d6	$7d7	$7d8	$7d9	$7da	$7db	$7dc	$7dd	$7de	$7df	$7e0	$7e1	$7e2	$7e3	$7e4	$7e5	$7e6	$7e7

That is, each character cell uses one byte of screen RAM, and the addresses increase smoothly, both within lines, and between lines. Super-Extended Attribute Mode requires two bytes per character cell. So if you set $D054 to $05, for example, you will get screen addresses like this:

$400	$402	$404	$406	$408	$40a	$40c	$40e	$410	$412	$414	$416	$418	$41a	$41c	$41e	$420	$422	$424	$426	$428	$42a	$42c	$42e	$430	$432	$434	$436	$438	$43a	$43c	$43e	$440	$442	$444	$446	$448	$44a	$44c	$44e
$428	$42a	$42c	$42e	$430	$432	$434	$436	$438	$43a	$43c	$43e	$440	$442	$444	$446	$448	$44a	$44c	$44e	$450	$452	$454	$456	$458	$45a	$45c	$45e	$460	$462	$464	$466	$468	$46a	$46c	$46e	$470	$472	$474	$476
$450	$452	$454	$456	$458	$45a	$45c	$45e	$460	$462	$464	$466	$468	$46a	$46c	$46e	$470	$472	$474	$476	$478	$47a	$47c	$47e	$480	$482	$484	$486	$488	$48a	$48c	$48e	$490	$492	$494	$496	$498	$49a	$49c	$49e
$478	$47a	$47c	$47e	$480	$482	$484	$486	$488	$48a	$48c	$48e	$490	$492	$494	$496	$498	$49a	$49c	$49e	$4a0	$4a2	$4a4	$4a6	$4a8	$4aa	$4ac	$4ae	$4b0	$4b2	$4b4	$4b6	$4b8	$4ba	$4bc	$4be	$4c0	$4c2	$4c4	$4c6
$4a0	$4a2	$4a4	$4a6	$4a8	$4aa	$4ac	$4ae	$4b0	$4b2	$4b4	$4b6	$4b8	$4ba	$4bc	$4be	$4c0	$4c2	$4c4	$4c6	$4c8	$4ca	$4cc	$4ce	$4d0	$4d2	$4d4	$4d6	$4d8	$4da	$4dc	$4de	$4e0	$4e2	$4e4	$4e6	$4e8	$4ea	$4ec	$4ee
$4c8	$4ca	$4cc	$4ce	$4d0	$4d2	$4d4	$4d6	$4d8	$4da	$4dc	$4de	$4e0	$4e2	$4e4	$4e6	$4e8	$4ea	$4ec	$4ee	$4f0	$4f2	$4f4	$4f6	$4f8	$4fa	$4fc	$4fe	$500	$502	$504	$506	$508	$50a	$50c	$50e	$510	$512	$514	$516
$4f0	$4f2	$4f4	$4f6	$4f8	$4fa	$4fc	$4fe	$500	$502	$504	$506	$508	$50a	$50c	$50e	$510	$512	$514	$516	$518	$51a	$51c	$51e	$520	$522	$524	$526	$528	$52a	$52c	$52e	$530	$532	$534	$536	$538	$53a	$53c	$53e
$518	$51a	$51c	$51e	$520	$522	$524	$526	$528	$52a	$52c	$52e	$530	$532	$534	$536	$538	$53a	$53c	$53e	$540	$542	$544	$546	$548	$54a	$54c	$54e	$550	$552	$554	$556	$558	$55a	$55c	$55e	$560	$562	$564	$566
$540	$542	$544	$546	$548	$54a	$54c	$54e	$550	$552	$554	$556	$558	$55a	$55c	$55e	$560	$562	$564	$566	$568	$56a	$56c	$56e	$570	$572	$574	$576	$578	$57a	$57c	$57e	$580	$582	$584	$586	$588	$58a	$58c	$58e
$568	$56a	$56c	$56e	$570	$572	$574	$576	$578	$57a	$57c	$57e	$580	$582	$584	$586	$588	$58a	$58c	$58e	$590	$592	$594	$596	$598	$59a	$59c	$59e	$5a0	$5a2	$5a4	$5a6	$5a8	$5aa	$5ac	$5ae	$5b0	$5b2	$5b4	$5b6
$590	$592	$594	$596	$598	$59a	$59c	$59e	$5a0	$5a2	$5a4	$5a6	$5a8	$5aa	$5ac	$5ae	$5b0	$5b2	$5b4	$5b6	$5b8	$5ba	$5bc	$5be	$5c0	$5c2	$5c4	$5c6	$5c8	$5ca	$5cc	$5ce	$5d0	$5d2	$5d4	$5d6	$5d8	$5da	$5dc	$5de
$5b8	$5ba	$5bc	$5be	$5c0	$5c2	$5c4	$5c6	$5c8	$5ca	$5cc	$5ce	$5d0	$5d2	$5d4	$5d6	$5d8	$5da	$5dc	$5de	$5e0	$5e2	$5e4	$5e6	$5e8	$5ea	$5ec	$5ee	$5f0	$5f2	$5f4	$5f6	$5f8	$5fa	$5fc	$5fe	$600	$602	$604	$606
$5e0	$5e2	$5e4	$5e6	$5e8	$5ea	$5ec	$5ee	$5f0	$5f2	$5f4	$5f6	$5f8	$5fa	$5fc	$5fe	$600	$602	$604	$606	$608	$60a	$60c	$60e	$610	$612	$614	$616	$618	$61a	$61c	$61e	$620	$622	$624	$626	$628	$62a	$62c	$62e
$608	$60a	$60c	$60e	$610	$612	$614	$616	$618	$61a	$61c	$61e	$620	$622	$624	$626	$628	$62a	$62c	$62e	$630	$632	$634	$636	$638	$63a	$63c	$63e	$640	$642	$644	$646	$648	$64a	$64c	$64e	$650	$652	$654	$656
$630	$632	$634	$636	$638	$63a	$63c	$63e	$640	$642	$644	$646	$648	$64a	$64c	$64e	$650	$652	$654	$656	$658	$65a	$65c	$65e	$660	$662	$664	$666	$668	$66a	$66c	$66e	$670	$672	$674	$676	$678	$67a	$67c	$67e
$658	$65a	$65c	$65e	$660	$662	$664	$666	$668	$66a	$66c	$66e	$670	$672	$674	$676	$678	$67a	$67c	$67e	$680	$682	$684	$686	$688	$68a	$68c	$68e	$690	$692	$694	$696	$698	$69a	$69c	$69e	$6a0	$6a2	$6a4	$6a6
$680	$682	$684	$686	$688	$68a	$68c	$68e	$690	$692	$694	$696	$698	$69a	$69c	$69e	$6a0	$6a2	$6a4	$6a6	$6a8	$6aa	$6ac	$6ae	$6b0	$6b2	$6b4	$6b6	$6b8	$6ba	$6bc	$6be	$6c0	$6c2	$6c4	$6c6	$6c8	$6ca	$6cc	$6ce
$6a8	$6aa	$6ac	$6ae	$6b0	$6b2	$6b4	$6b6	$6b8	$6ba	$6bc	$6be	$6c0	$6c2	$6c4	$6c6	$6c8	$6ca	$6cc	$6ce	$6d0	$6d2	$6d4	$6d6	$6d8	$6da	$6dc	$6de	$6e0	$6e2	$6e4	$6e6	$6e8	$6ea	$6ec	$6ee	$6f0	$6f2	$6f4	$6f6
$6d0	$6d2	$6d4	$6d6	$6d8	$6da	$6dc	$6de	$6e0	$6e2	$6e4	$6e6	$6e8	$6ea	$6ec	$6ee	$6f0	$6f2	$6f4	$6f6	$6f8	$6fa	$6fc	$6fe	$700	$702	$704	$706	$708	$70a	$70c	$70e	$710	$712	$714	$716	$718	$71a	$71c	$71e
$6f8	$6fa	$6fc	$6fe	$700	$702	$704	$706	$708	$70a	$70c	$70e	$710	$712	$714	$716	$718	$71a	$71c	$71e	$720	$722	$724	$726	$728	$72a	$72c	$72e	$730	$732	$734	$736	$738	$73a	$73c	$73e	$740	$742	$744	$746
$720	$722	$724	$726	$728	$72a	$72c	$72e	$730	$732	$734	$736	$738	$73a	$73c	$73e	$740	$742	$744	$746	$748	$74a	$74c	$74e	$750	$752	$754	$756	$758	$75a	$75c	$75e	$760	$762	$764	$766	$768	$76a	$76c	$76e
$748	$74a	$74c	$74e	$750	$752	$754	$756	$758	$75a	$75c	$75e	$760	$762	$764	$766	$768	$76a	$76c	$76e	$770	$772	$774	$776	$778	$77a	$77c	$77e	$780	$782	$784	$786	$788	$78a	$78c	$78e	$790	$792	$794	$796
$770	$772	$774	$776	$778	$77a	$77c	$77e	$780	$782	$784	$786	$788	$78a	$78c	$78e	$790	$792	$794	$796	$798	$79a	$79c	$79e	$7a0	$7a2	$7a4	$7a6	$7a8	$7aa	$7ac	$7ae	$7b0	$7b2	$7b4	$7b6	$7b8	$7ba	$7bc	$7be
$798	$79a	$79c	$79e	$7a0	$7a2	$7a4	$7a6	$7a8	$7aa	$7ac	$7ae	$7b0	$7b2	$7b4	$7b6	$7b8	$7ba	$7bc	$7be	$7c0	$7c2	$7c4	$7c6	$7c8	$7ca	$7cc	$7ce	$7d0	$7d2	$7d4	$7d6	$7d8	$7da	$7dc	$7de	$7e0	$7e2	$7e4	$7e6
$7c0	$7c2	$7c4	$7c6	$7c8	$7ca	$7cc	$7ce	$7d0	$7d2	$7d4	$7d6	$7d8	$7da	$7dc	$7de	$7e0	$7e2	$7e4	$7e6	$7e8	$7ea	$7ec	$7ee	$7f0	$7f2	$7f4	$7f6	$7f8	$7fa	$7fc	$7fe	$800	$802	$804	$806	$808	$80a	$80c	$80e

There are two things to notice in the above table: First, the address advances by two bytes for each character cell, because two bytes are required to define each character. Second, the start address of each screen line still only advances by 40 ($28 in hexadecimal). This isn't what we really want, because it means that half of the previous row will get displayed again on each current row. This is fixed by setting the number of bytes to advance each screen row in $D058 (LSB) and $D059 (MSB). So in this case, we want to increase the number of bytes skipped each line from 40 bytes, to 80 bytes, which we can do by setting $D058 to 80 ($50 in hexadecimal), and $D059 to 0. This gives us a screen layout like this:

$400	$402	$404	$406	$408	$40a	$40c	$40e	$410	$412	$414	$416	$418	$41a	$41c	$41e	$420	$422	$424	$426	$428	$42a	$42c	$42e	$430	$432	$434	$436	$438	$43a	$43c	$43e	$440	$442	$444	$446	$448	$44a	$44c	$44e
$450	$452	$454	$456	$458	$45a	$45c	$45e	$460	$462	$464	$466	$468	$46a	$46c	$46e	$470	$472	$474	$476	$478	$47a	$47c	$47e	$480	$482	$484	$486	$488	$48a	$48c	$48e	$490	$492	$494	$496	$498	$49a	$49c	$49e
$4a0	$4a2	$4a4	$4a6	$4a8	$4aa	$4ac	$4ae	$4b0	$4b2	$4b4	$4b6	$4b8	$4ba	$4bc	$4be	$4c0	$4c2	$4c4	$4c6	$4c8	$4ca	$4cc	$4ce	$4d0	$4d2	$4d4	$4d6	$4d8	$4da	$4dc	$4de	$4e0	$4e2	$4e4	$4e6	$4e8	$4ea	$4ec	$4ee
$4f0	$4f2	$4f4	$4f6	$4f8	$4fa	$4fc	$4fe	$500	$502	$504	$506	$508	$50a	$50c	$50e	$510	$512	$514	$516	$518	$51a	$51c	$51e	$520	$522	$524	$526	$528	$52a	$52c	$52e	$530	$532	$534	$536	$538	$53a	$53c	$53e
$540	$542	$544	$546	$548	$54a	$54c	$54e	$550	$552	$554	$556	$558	$55a	$55c	$55e	$560	$562	$564	$566	$568	$56a	$56c	$56e	$570	$572	$574	$576	$578	$57a	$57c	$57e	$580	$582	$584	$586	$588	$58a	$58c	$58e
$590	$592	$594	$596	$598	$59a	$59c	$59e	$5a0	$5a2	$5a4	$5a6	$5a8	$5aa	$5ac	$5ae	$5b0	$5b2	$5b4	$5b6	$5b8	$5ba	$5bc	$5be	$5c0	$5c2	$5c4	$5c6	$5c8	$5ca	$5cc	$5ce	$5d0	$5d2	$5d4	$5d6	$5d8	$5da	$5dc	$5de
$5e0	$5e2	$5e4	$5e6	$5e8	$5ea	$5ec	$5ee	$5f0	$5f2	$5f4	$5f6	$5f8	$5fa	$5fc	$5fe	$600	$602	$604	$606	$608	$60a	$60c	$60e	$610	$612	$614	$616	$618	$61a	$61c	$61e	$620	$622	$624	$626	$628	$62a	$62c	$62e
$630	$632	$634	$636	$638	$63a	$63c	$63e	$640	$642	$644	$646	$648	$64a	$64c	$64e	$650	$652	$654	$656	$658	$65a	$65c	$65e	$660	$662	$664	$666	$668	$66a	$66c	$66e	$670	$672	$674	$676	$678	$67a	$67c	$67e
$680	$682	$684	$686	$688	$68a	$68c	$68e	$690	$692	$694	$696	$698	$69a	$69c	$69e	$6a0	$6a2	$6a4	$6a6	$6a8	$6aa	$6ac	$6ae	$6b0	$6b2	$6b4	$6b6	$6b8	$6ba	$6bc	$6be	$6c0	$6c2	$6c4	$6c6	$6c8	$6ca	$6cc	$6ce
$6d0	$6d2	$6d4	$6d6	$6d8	$6da	$6dc	$6de	$6e0	$6e2	$6e4	$6e6	$6e8	$6ea	$6ec	$6ee	$6f0	$6f2	$6f4	$6f6	$6f8	$6fa	$6fc	$6fe	$700	$702	$704	$706	$708	$70a	$70c	$70e	$710	$712	$714	$716	$718	$71a	$71c	$71e
$720	$722	$724	$726	$728	$72a	$72c	$72e	$730	$732	$734	$736	$738	$73a	$73c	$73e	$740	$742	$744	$746	$748	$74a	$74c	$74e	$750	$752	$754	$756	$758	$75a	$75c	$75e	$760	$762	$764	$766	$768	$76a	$76c	$76e
$770	$772	$774	$776	$778	$77a	$77c	$77e	$780	$782	$784	$786	$788	$78a	$78c	$78e	$790	$792	$794	$796	$798	$79a	$79c	$79e	$7a0	$7a2	$7a4	$7a6	$7a8	$7aa	$7ac	$7ae	$7b0	$7b2	$7b4	$7b6	$7b8	$7ba	$7bc	$7be
$7c0	$7c2	$7c4	$7c6	$7c8	$7ca	$7cc	$7ce	$7d0	$7d2	$7d4	$7d6	$7d8	$7da	$7dc	$7de	$7e0	$7e2	$7e4	$7e6	$7e8	$7ea	$7ec	$7ee	$7f0	$7f2	$7f4	$7f6	$7f8	$7fa	$7fc	$7fe	$800	$802	$804	$806	$808	$80a	$80c	$80e
$810	$812	$814	$816	$818	$81a	$81c	$81e	$820	$822	$824	$826	$828	$82a	$82c	$82e	$830	$832	$834	$836	$838	$83a	$83c	$83e	$840	$842	$844	$846	$848	$84a	$84c	$84e	$850	$852	$854	$856	$858	$85a	$85c	$85e
$860	$862	$864	$866	$868	$86a	$86c	$86e	$870	$872	$874	$876	$878	$87a	$87c	$87e	$880	$882	$884	$886	$888	$88a	$88c	$88e	$890	$892	$894	$896	$898	$89a	$89c	$89e	$8a0	$8a2	$8a4	$8a6	$8a8	$8aa	$8ac	$8ae
$8b0	$8b2	$8b4	$8b6	$8b8	$8ba	$8bc	$8be	$8c0	$8c2	$8c4	$8c6	$8c8	$8ca	$8cc	$8ce	$8d0	$8d2	$8d4	$8d6	$8d8	$8da	$8dc	$8de	$8e0	$8e2	$8e4	$8e6	$8e8	$8ea	$8ec	$8ee	$8f0	$8f2	$8f4	$8f6	$8f8	$8fa	$8fc	$8fe
$900	$902	$904	$906	$908	$90a	$90c	$90e	$910	$912	$914	$916	$918	$91a	$91c	$91e	$920	$922	$924	$926	$928	$92a	$92c	$92e	$930	$932	$934	$936	$938	$93a	$93c	$93e	$940	$942	$944	$946	$948	$94a	$94c	$94e
$950	$952	$954	$956	$958	$95a	$95c	$95e	$960	$962	$964	$966	$968	$96a	$96c	$96e	$970	$972	$974	$976	$978	$97a	$97c	$97e	$980	$982	$984	$986	$988	$98a	$98c	$98e	$990	$992	$994	$996	$998	$99a	$99c	$99e
$9a0	$9a2	$9a4	$9a6	$9a8	$9aa	$9ac	$9ae	$9b0	$9b2	$9b4	$9b6	$9b8	$9ba	$9bc	$9be	$9c0	$9c2	$9c4	$9c6	$9c8	$9ca	$9cc	$9ce	$9d0	$9d2	$9d4	$9d6	$9d8	$9da	$9dc	$9de	$9e0	$9e2	$9e4	$9e6	$9e8	$9ea	$9ec	$9ee
$9f0	$9f2	$9f4	$9f6	$9f8	$9fa	$9fc	$9fe	$a00	$a02	$a04	$a06	$a08	$a0a	$a0c	$a0e	$a10	$a12	$a14	$a16	$a18	$a1a	$a1c	$a1e	$a20	$a22	$a24	$a26	$a28	$a2a	$a2c	$a2e	$a30	$a32	$a34	$a36	$a38	$a3a	$a3c	$a3e
$a40	$a42	$a44	$a46	$a48	$a4a	$a4c	$a4e	$a50	$a52	$a54	$a56	$a58	$a5a	$a5c	$a5e	$a60	$a62	$a64	$a66	$a68	$a6a	$a6c	$a6e	$a70	$a72	$a74	$a76	$a78	$a7a	$a7c	$a7e	$a80	$a82	$a84	$a86	$a88	$a8a	$a8c	$a8e
$a90	$a92	$a94	$a96	$a98	$a9a	$a9c	$a9e	$aa0	$aa2	$aa4	$aa6	$aa8	$aaa	$aac	$aae	$ab0	$ab2	$ab4	$ab6	$ab8	$aba	$abc	$abe	$ac0	$ac2	$ac4	$ac6	$ac8	$aca	$acc	$ace	$ad0	$ad2	$ad4	$ad6	$ad8	$ada	$adc	$ade
$ae0	$ae2	$ae4	$ae6	$ae8	$aea	$aec	$aee	$af0	$af2	$af4	$af6	$af8	$afa	$afc	$afe	$b00	$b02	$b04	$b06	$b08	$b0a	$b0c	$b0e	$b10	$b12	$b14	$b16	$b18	$b1a	$b1c	$b1e	$b20	$b22	$b24	$b26	$b28	$b2a	$b2c	$b2e
$b30	$b32	$b34	$b36	$b38	$b3a	$b3c	$b3e	$b40	$b42	$b44	$b46	$b48	$b4a	$b4c	$b4e	$b50	$b52	$b54	$b56	$b58	$b5a	$b5c	$b5e	$b60	$b62	$b64	$b66	$b68	$b6a	$b6c	$b6e	$b70	$b72	$b74	$b76	$b78	$b7a	$b7c	$b7e
$b80	$b82	$b84	$b86	$b88	$b8a	$b8c	$b8e	$b90	$b92	$b94	$b96	$b98	$b9a	$b9c	$b9e	$ba0	$ba2	$ba4	$ba6	$ba8	$baa	$bac	$bae	$bb0	$bb2	$bb4	$bb6	$bb8	$bba	$bbc	$bbe	$bc0	$bc2	$bc4	$bc6	$bc8	$bca	$bcc	$bce

It is possible to use Super-Extended Attribute Mode from C65-mode, by setting the screen to 80 columns, as the C65 ROM sets up 2KB for both the screen RAM and colour RAM, and this automatically sets $D058 and $D059 to the correct value for $40 \times 2 = 80$ bytes per screen line. The user need only to treat each character pair as a single Super-Extended Attribute character, and to enable Super-Extended Attribute Mode, as described above.

Because pairs of colour RAM and screen RAM bytes are used to define each character, care must be taken to initialise and manipulate the screen. A good approach is to set the text colour to black, because this is colour code 0, and then to fill the screen with @ characters, because that is character code 0. You can then have several ways to manipulate the screen. You can use the normal PRINT command and carefully construct strings that will put the correct values into each screen and colour byte pair. Another approach is to use the BANK and POKE commands to directly set the contents of the screen and colour RAM.

Managing a Super-Extended Attribute Mode screen in this way using BASIC 65 is of course rather a hack, and is only suggested as a relatively simple way to begin experimenting. You will almost certainly want to quickly move to using custom screen handling code, most probably in assembly, to manipulate Super-Extended Attribute Mode screens, although this approach of using BASIC 65 can be quite powerful, by allowing use of existing screen scrolling and other manipulations.

XXX Example program

The following descriptions assume that you have implemented one of the methods described above to set the screen and colour RAM.

Full-Colour (256 colours per character) Text Mode (FCM)

In normal VIC-II/III text mode, one byte is used for each row of pixels in a character. As a reminder for how those modes work, in hi-res mode, each pixel is either the background or foreground colour, based on the state of one bit in the byte. Multi-colour mode uses two bits to select between four possible colours, but as there are still only 8 bits to describe each row of 8 pixels, each pair of pixels has the same colour. The VIC-IV's full-colour text mode removes these limitations, and allows each pixel of a character to be chosen from the 256 colour of either the primary or alternate palette bank, without sacrificing horizontal resolution.

To do this, each character now requires 64 bytes of data. The address of the data is 64 $\times$ the character number, regardless of the character set address. FCM should normally be used with Super-Extended Attribute Mode (SEAM), so that more than 256 unique characters can be address. As SEAM allows the selection of 8,192 unique characters, this allows FCM character data to be placed anywhere in the first 512KB of chip RAM (but note that most models of the MEGA65 have only 384KB of chip RAM).

Please note that the pixel value $ff will not select the corresponding colour code directly. Instead, it will select the colour code defined by the colour RAM.

Nibble-colour (16 colours per character) Text Mode (NCM)

The Nibble-Colour Mode (NCM) for text is similar to Full-Colour Text Mode, except that each byte of data describes two pixels using 4 bits each. This makes the NCM unique, because the characters will be 16 pixels wide, instead of the usual 8 pixels wide. This can be used to create colourful displays, without using as much memory as FCM, because fewer characters are required to cover the screen. Unlike the VIC-II's MCM, this mode does not result in a loss of horizontal resolution.

In NCM the lower four bits of the pixel colour comes from the upper or lower four bits of the pixel data. The upper four bits of the colour code come from the colour RAM data for the displayed character. This makes it possible to use all palette entries in NCM, although the limitation of 16 colours per character remains. Similar to the behaviour of FCM, the pixel data value $f will select the pixel colour set in the colour RAM.

A further advantage of NCM is that it uses fewer bus cycles per pixel than FCM, because fewer character data fetches need to occur per raster line. Together with the reduced memory requirements, this makes NCM particularly useful for creating colourful multiple layers of graphics. This allows the VIC-IV to display arcade style displays with more colours than many 16-bit computers.

XXX

Alpha-Blending / Anti-Aliasing

The VIC-IV supports blending of characters with the background colour, enabling effects such as anti-aliased font rendering. Blending is possible on a per character basis. It is enabled for a specific character if the following conditions are met:

1. The *ALPHAEN* signal is set in bit 7 of $D054.
2. The *CHR16* signal is set in bit 0 of $D054 to enable Super-Extended Attribute Mode (SEAM).
3. Full-Colour Text Mode (FCM) is enabled for the character.

If alpha-blending is enabled for a character, its 8-bit pixel values are treated as alpha values instead of palette indices. The actual pixel colour is determined by blending the background colour with the character's foreground colour defined in the colour RAM. An alpha value of 0 represents full transparency, showing only the background colour for that pixel.

Note that the alpha-blending is only applied between the background colour and the character's foreground colour. This means that any characters behind the current character will effectively not be visible (character layers can be composed by using GOTOX repositioning). However, programmers should assume that blending with previous layers will be supported in a future implementation. To avoid issues with such a change you should not put any characters behind a character with alpha-blending enabled.

Flipping Characters

XXX

Variable Width Fonts

There are 4 bits that allow trimming pixels from the right edge of characters when they are displayed. This has the effect of making characters narrower. This can be useful for making more attractive text displays, where narrow characters, such as "i" take less space than wider characters, such as "m", without having to use a bitmap display. This feature can be used to make it very efficient to display such variable-width text displays - both in terms of memory usage and processing time.

This feature can be combined with full-colour text mode, alpha blending mode and 4-bits per pixel mode to allow characters that consist of 15 levels of intensity between the background and foreground colour, and that are up to 16 pixels wide. Further, the GOTO bit can be used to implement negative kerning, so that character pairs like A and T do not have excessive white space between them when printed adjacently. The prudent use of these features can result in highly impressive text display, similar to that on modern 32-bit and 64-bit systems, but that are still efficient enough to be implemented on a relatively constrained system such as the MEGA65. The "MegaWAT!?" presentation software for the MEGA65 uses several of these features to produce its attractive anti-aliased proportional text display on slides.

XXX MEGAWat!? screenshot

XXX Example program

Raster Re-write Buffer

If the GOTO bit is set for a character in Super-Extended Attribute Mode, instead of painting a character, the position on the raster is back-tracked (or advanced forward to) the pixel position specified in the low 10 bits of the screen memory bytes. If the vertical flip bit is set, then this has the alternate meaning of preventing the background colour from being painted. This combination can be used to print text material over the top of other text material, providing a crude supplement to the 8 hardware sprites. The amount of material is limited only by the raster time of the VIC-IV. Some experimentation will be required to determine how much can be achieved in PAL and NTSC modes.

If the GOTO bit is set for a character, and the character width reduction bits are also set, they are interpretted as a Y offset to add to the character data address, but only in Full Colour Mode. Setting Y=1 causes the character data to be fetched from 8 bytes later, i.e., the first row of character data will come from the address where the second row of character data would normally be fetched. Similary for increased values the character data will be fetched from further character rows. With careful arrangement of characters in memory, it is possible to use this feature to provide free vertical placement of soft sprites, without needing to copy the character data.

If the GOTO bit is set for a character, and the Nybl Colour Mode (NCM) bit is also set, then the second colour RAM byte for that character is used to set the Row Mask bits. For each bit set in the row mask, the corresponding row of characters in the line will not be displayed. This can be used in combination with the Y offset feature to effectively provide a character by character smooth vertical scrolling function.

This ability to draw multiple layers of text and graphics is highly powerful. For example, it can be used to provide multiple overlapping layers of separately scrollable graphics. This gives many of the advantages of bitplane-based play-fields on other computers, such as the Amiga, but without the disadvantages of bitplanes.

A good introduction to the Raster Re-write Buffer and its uses can be found in this video:

https://www.youtube.com/watch?v=00bm5uBeBos&feature=youtu.be

One important aspect of the RRB, is that the VIC-IV will display only the character data to the left of, and including, the last drawn character. This means that if you use the GOTO token to overwrite multiple layers of graphics, you must either make sure that the last layer reaches to the right-hand edge of the display, or you must include a GOTO token that moves the render position to the right-hand edge of the display.

XXX Example program

SPRITES

VIC-II/III Sprite Control

The control of sprites for C64 / VIC-II/III compatibility is unchanged from the C64. The only practical differences are very minor. In particular the VIC-IV uses ring-buffer for each sprites data when rendering a raster. This means that a sprite can be displayed multiple times per raster line, thus potentially allowing for horizontal multiplexing.

Extended Sprite Image Sets

On the VIC-II and VIC-III, all sprites must draw their image data from a single 16KB region of memory at any point in time. This limits the number of different sprite images to 256, because each sprite image occupies 64 bytes. In practice, the same 16KB region must also contain either bitmap, text or bitplane data, considerably reducing the number of sprite images that can be used at the same time.

The VIC-IV removes this limitation, by allowing sprite data to be placed anywhere in memory, although still on 64-byte boundaries. This is done by setting the SPRPTR16 signal (bit 7, $D06E, decimal 53358), which tells the VIC-IV to expect two bytes per sprite pointer instead of one. These addresses are then absolute addresses, and ignore the 16KB VIC-II bank selection logic. Thus 16 bytes are required instead of 8 bytes. The list of pointers can also be placed anywhere in memory by setting the SPRPTRADR ($D06C - $D06D, 53356 - 53357 decimal) and SPRPTRBNK signals (bits 0 - 6, $D06E, 53358 decimal). This allows for sprite data to be located anywhere in the first 4MB of RAM, and the sprite pointer list to be located anywhere in the first 8MB of RAM. Note that typical installations of the VIC-IV have only 384KB of connected RAM, so these limitations are of no practical effect. However, the upper bits of the SPRPTRBNK signal should be set to zero to avoid forward-compatibility problems.

One reason for supporting more sprite images is that sprites on the VIC-IV can require more than one 64 byte image slot. For example, enabling Extra-Wide Sprite Mode means that a sprite will require $8 \times 21 = 168$ bytes, and will thus occupy four VIC-II style 64 byte sprite image slots. If variable height sprites are used, this can grow to as much as $8 \times 255 = 2{,}040$ bytes per sprite.

Variable Sprite Size

Sprites can be one of three widths with the VIC-IV:

1. Normal VIC-II width (24 pixels wide).
2. Extra Wide, where 64 bits (8 bytes) of data are used per raster line, instead of the VIC-II's 24. This results in sprites that are 64 pixels wide, unless Full-Colour Sprite Mode is selected for a sprite, in which case the sprite will be 64 bits $\div$ 4 bits per pixel = 16 pixels wide.

3. Tiled mode, where the sprite is drawn repeatedly until the end of the raster line. Tiled mode should normally only be used with Extra Wide sprite mode, as the tiling always occurs using 64 bits of sprite data per line. Enabling tiled mode with normal 24 bit wide mono or multi-colour sprite data will draw 2 and 2/3 rows of sprite data as a single row, even if the given sprite is not in Extra Wide mode, resulting in garbled displays.

To enable a sprite to be 64 pixels (or 16 pixels if in Full-Colour Sprite Mode), set the corresponding bit for the sprite in the SPRX64EN register at ($D057, 53335 decimal). Enabling Full Colour mode for a sprite implicitly enables extended width mode, causes these sprites to be 16 pixels wide.

Similarly, sprites can be various heights: Sprites will be either the 21 pixels high of the VIC-II, or if the corresponding bit for the sprite is enabled in the SPRHGTEN signal ($D055, 53333 decimal), then that sprite will be the number of pixels tall that is set in the SPRHGT register ($D056, 53334 decimal), from 0 to 255. Notice that all sprites with SPRHGTEN enabled share the same height. A sprite can always leave the bottom of its image data transparent.

To enable tiled mode for a sprite, set the corresponding bit of SPRTILEN. For sprites 0 through 3, set bits 4 through 7 of $D04D (53325 decimal). For sprites 4 through 7, set bits 4 through 7 of $D04F (55327 decimal).

Variable Sprite Resolution

By default, sprites are the same resolution as on the VIC-II, i.e., each sprite pixel is two physical pixels wide and high. However, sprites can be made to use the native resolution, where sprite pixels are one physical pixel wide and/or high. This is achieved by setting the relevant bit for the sprite in the SPRENV400 ($D076, 53366 decimal) registers to increase the vertical resolution on a sprite-by-sprite basis. The horizontal resolution for all sprites is either the normal VIC-II resolution, or if the SPR640 signal is set (bit 4 of $D054, 53332 decimal), then sprites will have the same horizontal resolution as the physical pixels of the display.

Sprite Palette Bank

The VIC-IV has four palette banks, compared with the single palette bank of the VIC-III. The VIC-IV allows the selection of separate palette banks for bitmap/text graphics and for sprites. This makes it easy to have very colourful displays, where the sprites have different colours to the rest of the display, or to use palette animation to achieve interesting visual effects in sprites, without disturbing the palette used by other elements of the display.

The sprite palette bank is selected by setting the SPRPALSEL signal in bits 2 and 3 of the register $D070 (53360 decimal). It is possible to set this to the same bank as the bitmap/text display, or to select a different palette bank. Palette bank selection takes effect immediately. Don't forget that to be able to modify a palette, you have to also bank it to be the palette accessible via the palette bank registers at $D100 - $D3FF by setting the MAPEDPAL signal in bits 6 and 7 of $D070.

Full-Colour Sprite Mode

In addition to monochrome and multi-colour modes, the VIC-IV supports a new full-colour sprite mode. In this mode, four bits are used to encode each sprite pixel. However, unlike multi-colour mode where pairs of bits encode pairs of pixels, in full-colour mode the pixels remain at their normal horizontal resolution. The colour zero is considered transparent. If you wish to use black in a full-colour sprite, you must configure the palette bank that is selected for sprites so that one of the 15 colours for the specific sprite encodes black.

Full-colour sprite mode is selectable for each sprite by setting the appropriate bit in the SPR16EN register ($D06B, 53355 decimal).

To enable the eight sprites to have 15 unique colours each, the sprite colour is drawn using the palette entry corresponding to: $spritenumber \times 16 + nibblevalue$, where $spritenumber$ is the number of the sprite (from 0 to 7), and $nibblevalue$ is the value of the half-byte that contains the sprite data for the pixel. In addition, if bitplane mode is enabled for this sprite, then 128 is added to the colour value, which makes it easy to switch between two colour schemes for a given sprite by changing only one bit in the SPRBPMEN register.

Because Full-Colour Sprite Mode requires four bits per pixel, sprites will be only six pixels wide, unless Extra Wide Sprite Mode is enabled for a sprite, in which case the sprite will be 16 pixels wide. Tiled Mode also works with Full-Colour Sprite Mode, and will result in the 16 full-colour pixels of the sprite being repeated until the end of the raster line.

The following BASIC program draws a Full-Colour Sprite in either C64 or C65-mode:

```
10 PRINT CHR$(147)
20 REM C65/C64-MODE DETECT
30 IF PEEK(53272) AND 32 THEN GOTO 100
40 POKE 53295,ASC("G"):POKE 53295,ASC("S")
100 REM SETUP SPRITE
110 AD=4096                          :REM $1000 SPRITE ADDR
120 TC=10                            :REM TRANSPARENT COLOUR
130 SPR=PEEK(53356)+PEEK(53357)*256 :REM GET SPRITE TABLE ADDRESS
140 POKE SPR,AD/64                   :REM SET SPRITE ADDRESS
150 FOR I=AD TO AD+168               :REM CLEAR SPRITE WITH TC
160 POKE I,TC+TC*16                  :REM ONE BYTE = 2 PIXEL
170 NEXT
180 POKE 53287,TC                    :REM SET TRANSPARENT COLOUR
190 POKE 53248,100                   :REM PUT SPRITE...
200 POKE 53249,100                   :REM ON SCREEN AT 100,100
210 POKE 53355,1                     :REM MAKE SPRITE 0 16-COLOUR
220 POKE 53335,1                     :REM MAKE SPRITE 0 USE 16X4-BITS
230 POKE 53269,1                     :REM ENABLE SPRITE 0
240 GOSUB 900                        :REM READ MULTI-COLOUR SPRITE
250 END

900 REM LOAD SPRITE FROM DATA
910 READ N$:IF N$="END" THEN RETURN
920 GOSUB 1000                       :REM DECODE LINE
930 GOTO 910
```

```
1000 REM DECODE STRING OF NIBBLES IN N$ AT ADDRESS AD
1010 IF LEN(N$)<>16 THEN PRINT "ILLEGAL SPR DATA!":END
1020 FOR I=1 TO 16 STEP 2
1030 N=(ASC(MID$(N$,I,1))-ASC("@"))    :REM HIGH NYB
1040 IF N<0 THEN N=TC                  :REM . IS TRANSPARENT
1050 M=(ASC(MID$(N$,I+1,1))-ASC("@"))  :REM LOW NYB
1060 IF M<0 THEN M=TC                  :REM . IS TRANSPARENT
1070 POKE AD,(N AND 15)*16 + (M AND 15):REM SET 2 PIXELS
1080 AD=AD+1                           :REM ADVANCE AD
1090 NEXT I
1100 RETURN

1998 REM SPRITE DATA
1999 REM . = TRANSPARENT, @-O = COLOURS 0 TO 15
2000 DATA "..AAFF...HHCC..."
2010 DATA ".AAFF.....HHCC.."
2020 DATA "AAFF.......HHCC."
2030 DATA "AFF...@@@...HHC."
2040 DATA "FF..@@@@@@..HH."
2050 DATA "..@@@@@@@@@@..."
2060 DATA ".@@@@@@@@@@@@.."
2070 DATA ".@@@@@@@@@@@@.."
2080 DATA "@@@@@@@@@@@@@@@."
2090 DATA "@@@@@@@@@@@@@@@."
2100 DATA "@@@@@@@@@@@@@@@."
2110 DATA "@@@@@@BB@BB@@@@@."
2120 DATA "@@@@BBBBBBBBB@@@."
2130 DATA ".@@@@@BBBBB@@@@.."
2140 DATA ".@@@@@@BB@@@@@.."
2150 DATA "..@@@@@@@@@@@..."
2160 DATA "II..@@@@@@..KK."
2170 DATA "DII...@@@...KKE."
2180 DATA "DDII.......KKEE."
2190 DATA ".DDII.....KKEE.."
2200 DATA "..DDII...KKEE..."
2210 DATA "END"
```

VIC-III ERRATA LEVEL

There are a few cases where the VIC-III chip in the Commodore 65 prototypes that are known to exist either do not behave as specified, the specification lacks detail and the implementation is oddly inconsistent, or the design itself has flaws or inconsistencies. The default behavior of the VIC-IV is to emulate the VIC-III as closely as possible in these cases. In some cases where the VIC-III behavior is lacking, the VIC-IV provides improved behaviors that can be selected by software using the HWERRATA register at $D08F.

Because these fixes are backwards incompatible with the VIC-III and with earlier versions of the VIC-IV and the MEGA65 core, software must opt in to these fixes by setting the HWERRATA register. This protects software from further changes that may be introduced in future versions of the MEGA65 core. The boot state of this register is $00, which requests full compatibility with the VIC-III, including buggy behaviors. The MEGA65 ROM will always leave this set to $00 when launching programs. A program can set HWERRATA to enable a set of fixes known to the developer at the time the program is written, and exclude backwards incompatible fixes that might be introduced in later versions.

Requesting an errata level enables all fixes up to that level. By design, there is no way to request a level and exclude specific fixes at lower levels. You must write your program to accommodate all fixes up to the requested level.

In cases where enabling a fix changes the behavior of a hot register, setting the errata level does *not* trigger hot register propagation. The program must trigger hot register propagation by writing a 1 to the HOTREG register.

The errata levels implemented so far are as follows:

VIC-III Errata Levels

Level	Fixed behavior	Release introduced
0	No fixes. Fully VIC-III compatible.	N/A
1	**X scroll position shifted in H640 mode.** The VIC-III renders the text smooth scroll X position ($D016 XSCL) incorrectly in H640 mode. The fix offsets the scroll position 1 logical pixel (2 physical pixels) to the right. This does not take effect until hot register propagation.	v0.96

continued ...

Level	Fixed behavior	Release introduced
2	**Character attribute combinations.** When the upper palette ("bold") character attribute (bit 6) is set, the VIC-III has counterintuitive behaviors when "blink" (bit 4) or "reverse" (bit 5) are also set: blink will toggle the upper palette attribute and not blink the character, and reverse has no effect. With this fix, upper palette + blink will blink the character, and bold + reverse will reverse the character, displayed with the upper palette in both cases.	v0.96
3	**SD Card Busy Flag behaviour.** The SD card busy flag (bit 1 of $D680) indicates if the low-level SD card controller is busy. The addition of the SD card controller read-ahead functionality means that older software that expectes this bit to clear on completion of a read operation will incorrectly wait for the entire read-ahead sequence to complete. This may in turn result in software incorrectly believing the sector read has failed due to the longer time required, or that another read request cannot be immediately submitted. Therefore below this errata level the SD card busy flag does not indicate if the SD card controller is performing a background sector read-ahead operation. At or above this errata level this information is not concealed, i.e., bit 0 will clear when the requested sector is available, but bit 1 will remain set while any read-ahead operation continues.	

VIC-II / C64 REGISTERS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D000	53248	S0X							
D001	53249	S0Y							
D002	53250	S1X							
D003	53251	S1Y							
D004	53252	S2X							
D005	53253	S2Y							
D006	53254	S3X							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D007	53255	S3Y							
D008	53256	S4X							
D009	53257	S4Y							
D00A	53258	S5X							
D00B	53259	S5Y							
D00C	53260	S6X							
D00D	53261	S6Y							
D00E	53262	S7X							
D00F	53263	S7Y							
D010	53264	SXMSB							
D011	53265	RC8	ECM	BMM	BLNK	RSEL	YSCL		
D012	53266	RC							
D013	53267	LPX							
D014	53268	LPY							
D015	53269	SE							
D016	53270	-		RST	MCM	CSEL	XSCL		
D017	53271	SEXY							
D018	53272	VS				CB			-
D019	53273	-				ILP	ISSC	ISBC	RIRQ
D01A	53274	-					MISSC	MISBC	MRIRQ
D01B	53275	BSP							
D01C	53276	SCM							
D01D	53277	SEXX							
D01E	53278	SSC							
D01F	53279	SBC							
D020	53280	-				BORDERCOL			
D021	53281	-				SCREENCOL			
D022	53282	-				MC1			
D023	53283	-				MC2			
D024	53284	-				MC3			
D025	53285	SPRMC0							
D026	53286	SPRMC1							
D027	53287	SPR0COL							
D028	53288	SPR1COL							
D029	53289	SPR2COL							
D02A	53290	SPR3COL							
D02B	53291	SPR4COL							
D02C	53292	SPR5COL							
D02D	53293	SPR6COL							
D02E	53294	SPR7COL							
D030	53296	-							C128-FAST

- **BLNK** Enable display: 0 = blank the display, 1 = show the display
- **BMM** bitmap mode
- **BORDERCOL** display border colour (16 colour)
- **BSP** sprite background priority bits
- **C128FAST** 2MHz select (for C128 2MHz emulation)
- **CB** character set address location (× 1KiB)
- **CSEL** 38/40 column select
- **ECM** extended background mode
- **ILP** light pen indicate or acknowledge
- **ISBC** sprite:bitmap collision indicate or acknowledge
- **ISSC** sprite:sprite collision indicate or acknowledge
- **LPX** Coarse horizontal beam position (was lightpen X)
- **LPY** Coarse vertical beam position (was lightpen Y)
- **MC1** multi-colour 1 (16 colour)
- **MC2** multi-colour 2 (16 colour)
- **MC3** multi-colour 3 (16 colour)
- **MCM** Multi-colour mode
- **MISBC** mask sprite:bitmap collision IRQ
- **MISSC** mask sprite:sprite collision IRQ
- **MRIRQ** mask raster IRQ
- **RC** raster compare bits 0 to 7
- **RC8** raster compare bit 8
- **RIRQ** raster compare indicate or acknowledge
- **RSEL** 24/25 row select
- **RST** Disables video output on MAX Machine(tm) VIC-II 6566. Ignored on normal C64s and the MEGA65
- **SBC** sprite/foreground collision indicate bits
- **SCM** sprite multicolour enable bits
- **SCREENCOL** screen colour (16 colour)
- **SE** sprite enable bits
- **SEXX** sprite horizontal expansion enable bits

- **SEXY** sprite vertical expansion enable bits
- **SNX** sprite N horizontal position
- **SNY** sprite N vertical position
- **SPRMC0** Sprite multi-colour 0
- **SPRMC1** Sprite multi-colour 1
- **SPRNCOL** sprite N colour / 16-colour sprite transparency colour (lower nybl)
- **SSC** sprite/sprite collision indicate bits
- **SXMSB** sprite horizontal position MSBs
- **VS** screen address (× 1KiB)
- **XSCL** horizontal smooth scroll
- **YSCL** 24/25 vertical smooth scroll

VIC-III / C65 REGISTERS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D020	53280	BORDERCOL							
D021	53281	SCREENCOL							
D022	53282	MC1							
D023	53283	MC2							
D024	53284	MC3							
D025	53285	SPRMC0							
D026	53286	SPRMC1							
D02F	53295	KEY							
D030	53296	ROME	CROM9	ROMC	ROMA	ROM8	PAL	EXTSYNC	CRAM2K
D031	53297	H640	FAST	ATTR	BPM	V400	H1280	MONO	INT
D033	53299	B0ADODD			-	B0ADEVN			-
D034	53300	B1ADODD			-	B1ADEVN			-
D035	53301	B2ADODD			-	B2ADEVN			-
D036	53302	B3ADODD			-	B3ADEVN			-
D037	53303	B4ADODD			-	B4ADEVN			-
D038	53304	B5ADODD			-	B5ADEVN			-
D039	53305	B6ADODD			-	B6ADEVN			-
D03A	53306	B7ADODD			-	B7ADEVN			-
D03B	53307	BPCOMP							
D03C	53308	BPX							
D03D	53309	BPY							
D03E	53310	HPOS							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D03F	53311	VPOS							
D040	53312	B0PIX							
D041	53313	B1PIX							
D042	53314	B2PIX							
D043	53315	B3PIX							
D044	53316	B4PIX							
D045	53317	B5PIX							
D046	53318	B6PIX							
D047	53319	B7PIX							
D100 - D1FF	53504 - 53759	PALRED							
D200 - D2FF	53760 - 54015	PALGREEN							
D300 - D3FF	54016 - 54271	PALBLUE							

- **ATTR** Enable extended attributes and 8 bit colour entries
- **BNPIX** Display Address Translater (DAT) Bitplane N port
- **BORDERCOL** display border colour (256 colour)
- **BPCOMP** Complement bitplane flags
- **BPM** Bit-Plane Mode
- **BPX** Bitplane X
- **BPY** Bitplane Y
- **BXADEVN** Bitplane X address, even lines
- **BXADODD** Bitplane X address, odd lines
- **CRAM2K** Map 2nd KB of colour RAM $DC00-$DFFF
- **CROM9** Select between C64 and C65 charset.
- **EXTSYNC** Enable external video sync (genlock input)
- **FAST** Enable C65 FAST mode (~3.5MHz)
- **H1280** Enable 1280 horizontal pixels (not implemented)
- **H640** Enable C64 640 horizontal pixels / 80 column mode
- **HPOS** Bitplane X Offset
- **INT** Enable VIC-III interlaced mode
- **KEY** Write $A5 then $96 to enable C65/VIC-III IO registers

- **MC1** multi-colour 1 (256 colour)
- **MC2** multi-colour 2 (256 colour)
- **MC3** multi-colour 3 (256 colour)
- **MONO** Enable VIC-III MONO composite video output (colour if disabled)
- **PAL** Use PALETTE ROM (0) or RAM (1) entries for colours 0 - 15
- **PALBLUE** blue palette values (reversed nybl order)
- **PALGREEN** green palette values (reversed nybl order)
- **PALRED** red palette values (reversed nybl order)
- **ROM8** Map C65 ROM $8000
- **ROMA** Map C65 ROM $A000
- **ROMC** Map C65 ROM $C000
- **ROME** Map C65 ROM $E000
- **SCREENCOL** screen colour (256 colour)
- **SPRMC0** Sprite multi-colour 0 (8-bit for selection of any palette colour)
- **SPRMC1** Sprite multi-colour 1 (8-bit for selection of any palette colour)
- **V400** Enable 400 vertical pixels
- **VPOS** Bitplane Y Offset

VIC-IV / MEGA65 SPECIFIC REGISTERS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D020	53280	BORDERCOL							
D021	53281	SCREENCOL							
D022	53282	MC1							
D023	53283	MC2							
D024	53284	MC3							
D025	53285	SPRMC0							
D026	53286	SPRMC1							
D02F	53295	KEY							
D048	53320	TBDRPOS							
D049	53321	SPRBPMEN				TBDRPOS			
D04A	53322	BBDRPOS							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D04B	53323	SPRBPMEN				BBDRPOS			
D04C	53324	TEXTXPOS							
D04D	53325	SPRTILEN				TEXTXPOS			
D04E	53326	TEXTYPOS							
D04F	53327	SPRTILEN				TEXTYPOS			
D050	53328	XPOSLSB							
D051	53329	NORRDEL	DBLRR	XPOSMSB					
D052	53330	FNRASTERLSB							
D053	53331	FNRST	SHDEMU	UPSCALE	RE-SERVED	-	FNRASTERMSB		
D054	53332	ALPHEN	VFAST	PALEMU	SPRH640	SMTH	FCLRHI	FCLRLO	CHR16
D055	53333	SPRHGTEN							
D056	53334	SPRHGHT							
D057	53335	SPRX64EN							
D058	53336	LINESTEPLSB							
D059	53337	LINESTEPMSB							
D05A	53338	CHRXSCL							
D05B	53339	CHRYSCL							
D05C	53340	SDBDRWDLSB							
D05D	53341	HOTREG	RSTDELEN	SDBDRWDMSB					
D05E	53342	CHRCOUNT							
D05F	53343	SPRXSMSBS							
D060	53344	SCRNPTRLSB							
D061	53345	SCRNPTRMSB							
D062	53346	SCRNPTRBNK							
D063	53347	EXGLYPH	FCOLMCM	CHRCOUNT		SCRNPTRMB			
D064	53348	COLPTRLSB							
D065	53349	COLPTRMSB							
D068	53352	CHARPTRLSB							
D069	53353	CHARPTRMSB							
D06A	53354	CHARPTRBNK							
D06B	53355	SPR16EN							
D06C	53356	SPRPTRADRLSB							
D06D	53357	SPRPTRADRMSB							
D06E	53358	SPRPTR16	SPRPTRBNK						
D06F	53359	PALNTSC	VGAHDTV	RASLINE0					
D070	53360	MAPEDPAL		BTPALSEL		SPRPALSEL		ABTPALSEL	
D071	53361	BP16ENS							
D072	53362	SPRYADJ							
D073	53363	RASTERHEIGHT				ALPHADELAY			
D074	53364	SPRENALPHA							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D075	53365	SPRALPHAVAL							
D076	53366	SPRENV400							
D077	53367	SPRYMSBS							
D078	53368	SPRYSMSBS							
D079	53369	RASCMP							
D07A	53370	FNRST-CMP	EXTIRQS	NOBUG-COMPAT	CHARY16	SPTR-CONT	RASCMPMSB		
D07B	53371	DISPROWS							
D07C	53372	DEBUGC		VSYNCP	HSYNCP	RESV	BITPBANK		

- **ABTPALSEL** VIC-IV bitmap/text palette bank (alternate palette)
- **ALPHADELAY** Alpha delay for compositor
- **ALPHEN** Alpha compositor enable
- **BBDRPOS** bottom border position
- **BITPBANK** Set which 128KB bank bitplanes
- **BORDERCOL** display border colour (256 colour)
- **BP16ENS** VIC-IV 16-colour bitplane enable flags
- **BTPALSEL** bitmap/text palette bank
- **CHARPTRBNK** Character set precise base address (bits 23 - 16)
- **CHARPTRLSB** Character set precise base address (bits 0 - 7)
- **CHARPTRMSB** Character set precise base address (bits 15 - 8)
- **CHARY16** Alternate char ROM bank on alternate raster lines in V200
- **CHR16** enable 16-bit character numbers (two screen bytes per character)
- **CHRCOUNT** Number of characters to display per row (LSB)
- **CHRXSCL** Horizontal hardware scale of text mode (pixel 120ths per pixel)
- **CHRYSCL** Vertical scaling of text mode (number of physical rasters per char text row)
- **COLPTRLSB** colour RAM base address (bits 0 - 7)
- **COLPTRMSB** colour RAM base address (bits 15 - 8)
- **DBLRR** When set, the Raster Rewrite Buffer is only updated every 2nd raster line, limiting resolution to V200, but allowing more cycles for Raster-Rewrite actions.
- **DEBUGC** VIC-IV debug pixel select red(01), green(10) or blue(11) channel visible in $D07D

- **DISPROWS** Number of text rows to display
- **EXGLYPH** source full-colour character data from expansion RAM
- **EXTIRQS** Enable additional IRQ sources, e.g., raster X position.
- **FCLRHI** enable full-colour mode for character numbers >$FF
- **FCLRLO** enable full-colour mode for character numbers <=$FF
- **FCOLMCM** enable 256 colours in multicolour text mode
- **FNRASTERLSB** Read physical raster position
- **FNRASTERMSB** Read physical raster position
- **FNRST** Read raster compare source (0=VIC-IV fine raster, 1=VIC-II raster), provides same value as set in FNRSTCMP
- **FNRSTCMP** Raster compare is in physical rasters if clear, or VIC-II rasters if set
- **HOTREG** Enable VIC-II hot registers. When enabled, touching many VIC-II registers causes the VIC-IV to recalculate display parameters, such as border positions and sizes. Touching registers while this is disabled will trigger a change when reenabling. Setting this to 0 will clear the recalc flag, canceling the recalculation.
- **HSYNCP** hsync polarity
- **KEY** Write $47 then $53 to enable C65GS/VIC-IV IO registers
- **LINESTEPLSB** number of bytes to advance between each text row (LSB)
- **LINESTEPMSB** number of bytes to advance between each text row (MSB)
- **MAPEDPAL** palette bank mapped at $D100-$D3FF
- **MC1** multi-colour 1 (256 colour)
- **MC2** multi-colour 2 (256 colour)
- **MC3** multi-colour 3 (256 colour)
- **NOBUGCOMPAT** Disables VIC-III / C65 Bug Compatibility Mode if set
- **NORRDEL** When clear, raster rewrite double buffering is used
- **PALEMU** Enable PAL CRT-like scan-line emulation
- **PALNTSC** NTSC emulation mode (max raster = 262)
- **RASCMP** Physical raster compare value to be used if FNRSTCMP is clear
- **RASCMPMSB** Raster compare value MSB
- **RASLINE0** first VIC-II raster line
- **RASTERHEIGHT** physical rasters per VIC-II raster (1 to 16)

- **RESERVED** Reserved
- **RSTDELEN** Enable raster delay (delays raster counter and interrupts by one line to match output pipeline latency)
- **SCREENCOL** screen colour (256 colour)
- **SCRNPTRBNK** screen RAM precise base address (bits 23 - 16)
- **SCRNPTRLSB** screen RAM precise base address (bits 0 - 7)
- **SCRNPTRMB** screen RAM precise base address (bits 31 - 24)
- **SCRNPTRMSB** screen RAM precise base address (bits 15 - 8)
- **SDBDRWDLSB** Width of single side border (LSB)
- **SDBDRWDMSB** side border width (MSB)
- **SHDEMU** Enable simulated shadow-mask (PALEMU must also be enabled)
- **SMTH** video output horizontal smoothing enable
- **SPR16EN** sprite 16-colour mode enables
- **SPRALPHAVAL** Sprite alpha-blend value
- **SPRBPMEN** Sprite bitplane-modify-mode enables
- **SPRENALPHA** Sprite alpha-blend enable
- **SPRENV400** Sprite V400 enables
- **SPRH640** Sprite H640 enable
- **SPRHGHT** Sprite extended height size (sprite pixels high)
- **SPRHGTEN** sprite extended height enable (one bit per sprite)
- **SPRMC0** Sprite multi-colour 0 (8-bit for selection of any palette colour)
- **SPRMC1** Sprite multi-colour 1 (8-bit for selection of any palette colour)
- **SPRPALSEL** sprite palette bank
- **SPRPTR16** 16-bit sprite pointer mode (allows sprites to be located on any 64 byte boundary in chip RAM)
- **SPRPTRADRLSB** sprite pointer address (bits 7 - 0)
- **SPRPTRADRMSB** sprite pointer address (bits 15 - 8)
- **SPRPTRBNK** sprite pointer address (bits 23 - 16)
- **SPRTILEN** Sprite horizontal tile enables.
- **SPRX64EN** Sprite extended width enables (8 bytes per sprite row = 64 pixels wide for normal sprites or 16 pixels wide for 16-colour sprite mode)

- **SPRXSMSBS** Sprite H640 X Super-MSBs
- **SPRYADJ** Sprite Y position adjustment
- **SPRYMSBS** Sprite V400 Y position MSBs
- **SPRYSMSBS** Sprite V400 Y position super MSBs
- **SPTRCONT** Continuously monitor sprite pointer, to allow changing sprite data source while a sprite is being drawn
- **TBDRPOS** top border position
- **TEXTXPOS** character generator horizontal position
- **TEXTYPOS** Character generator vertical position
- **UPSCALE** Enable integrated low-latency (130usec) 720p upscaler
- **VFAST** C65GS FAST mode (48MHz)
- **VGAHDTV** Select more VGA-compatible mode if set, instead of HDMI/HDTV VIC-II cycle-exact frame timing. May help to produce a functional display on older VGA monitors.
- **VSYNCP** vsync polarity
- **XPOSLSB** Read horizontal raster scan position LSB
- **XPOSMSB** Read horizontal raster scan position MSB

CHAPTER 3

Sound Interface Device (SID)

- SID Registers

SID REGISTERS

The MEGA65 has 4 SIDs build in, which can be access through the register ranges starting at $D400, $D420, $D440, and $D460. The registers in each of these ranges are exactly the same, so the following table only lists the first SID. Add 32 the get to the next SID respectively.

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D400	54272	VOICE1FRQLO							
D401	54273	VOICE1FRQHI							
D402	54274	VOICE1PWLO							
D403	54275	VOICE1UNSD				VOICE1PWHI			
D404	54276	VOICE1-CTRLRNW	VOICE1-CTRLPUL	VOICE1-CTRLSAW	VOICE1-CTRLTRI	VOICE1-CTRLTST	VOICE1-CTRLRMO	VOICE1-CTRLRMF	VOICE1-CTRLGATE
D405	54277	ENV1ATTDUR				ENV1DECDUR			
D406	54278	ENV1SUSDUR				ENV1RELDUR			
D407	54279	VOICE2FRQLO							
D408	54280	VOICE2FRQHI							
D409	54281	VOICE2PWLO							
D40A	54282	VOICE2UNSD				VOICE2PWHI			
D40B	54283	VOICE2-CTRLRNW	VOICE2-CTRLPUL	VOICE2-CTRLSAW	VOICE2-CTRLTRI	VOICE2-CTRLTST	VOICE2-CTRLRMO	VOICE2-CTRLRMF	VOICE2-CTRLGATE
D40C	54284	ENV2ATTDUR				ENV2DECDUR			
D40D	54285	ENV2SUSDUR				ENV2RELDUR			
D40E	54286	VOICE3FRQLO							
D40F	54287	VOICE3FRQHI							
D410	54288	VOICE3PWLO							
D411	54289	VOICE3UNSD				VOICE3PWHI			
D412	54290	VOICE3-CTRLRNW	VOICE3-CTRLPUL	VOICE3-CTRLSAW	VOICE3-CTRLTRI	VOICE3-CTRLTST	VOICE3-CTRLRMO	VOICE3-CTRLRMF	VOICE3-CTRLGATE
D413	54291	ENV3ATTDUR				ENV3DECDUR			
D414	54292	ENV3SUSDUR				ENV3RELDUR			
D415	54293	FLTRCUTFRQLO							
D416	54294	FLTRCUTFRQHI							
D417	54295	FLTRRESON				FLTR-EXTINP	FLTR-V1OUT	FLTR-V2OUT	FLTR-V3OUT
D418	54296	FLTR-CUTV3	FLTR-HIPASS	FLTR-BDPASS	FLTR-LOPASS	FLTRVOL			
D419	54297	PADDLE1							
D41A	54298	PADDLE2							
D41B	54299	OSC3RNG							
D41C	54300	ENV3OUT							
D63C	54844	-				SIDMODE			

- **ENV3OUT** Envelope Generator 3 Output

- **ENVXATTDUR** Envelope Generator X Attack Cycle Duration
- **ENVXDECDUR** Envelope Generator X Decay Cycle Duration
- **ENVXRELDUR** Envelope Generator X Release Cycle Duration
- **ENVXSUSDUR** Envelope Generator X Sustain Cycle Duration
- **FLTRBDPASS** Filter Band-Pass Mode
- **FLTRCUTFRQHI** Filter Cutoff Frequency High
- **FLTRCUTFRQLO** Filter Cutoff Frequency Low
- **FLTRCUTV3** Filter Cut-Off Voice 3 Output (1 = off)
- **FLTREXTINP** Filter External Input
- **FLTRHIPASS** Filter High-Pass Mode
- **FLTRLOPASS** Filter Low-Pass Mode
- **FLTRRESON** Filter Resonance
- **FLTRVOL** Filter Output Volume
- **FLTRVXOUT** Filter Voice X Output
- **OSC3RNG** Oscillator 3 Random Number Generator
- **PADDLE1** Analog/Digital Converter: Game Paddle 1 (0-255)
- **PADDLE2** Analog/Digital Converter Game Paddle 2 (0-255)
- **SIDMODE** Select SID mode: 0=6581, 1=8580
- **VOICE1CTRLRMF** Voice 1 Synchronize Osc. 1 with Osc. 3 Frequency
- **VOICE1CTRLRMO** Voice 1 Ring Modulate Osc. 1 with Osc. 3 Output
- **VOICE2CTRLRMF** Voice 2 Synchronize Osc. 2 with Osc. 1 Frequency
- **VOICE2CTRLRMO** Voice 2 Ring Modulate Osc. 2 with Osc. 1 Output
- **VOICE3CTRLRMF** Voice 3 Synchronize Osc. 3 with Osc. 2 Frequency
- **VOICE3CTRLRMO** Voice 3 Ring Modulate Osc. 3 with Osc. 2 Output
- **VOICEXCTRLGATE** Voice X Gate Bit (1 = Start, 0 = Release)
- **VOICEXCTRLPUL** Voice X Pulse Waveform
- **VOICEXCTRLRNW** Voice X Control Random Noise Waveform
- **VOICEXCTRLSAW** Voice X Sawtooth Waveform
- **VOICEXCTRLTRI** Voice X Triangle Waveform
- **VOICEXCTRLTST** Voice X Test Bit - Disable Oscillator

- **VOICEXFRQHI** Voice X Frequency High
- **VOICEXFRQLO** Voice X Frequency Low
- **VOICEXPWHI** Voice X Pulse Waveform Width High
- **VOICEXPWLO** Voice X Pulse Waveform Width Low
- **VOICEXUNSD** Unused

CHAPTER 4

F018-Compatible Direct Memory Access (DMA) Controller

- F018A/B DMA Jobs
- MEGA65 Enhanced DMA Jobs
- Texture Scaling and Line Drawing
- Inline DMA Lists
- Audio DMA
- F018 “DMAgic” DMA Controller
- MEGA65 DMA Controller Extensions
- Unimplemented Functionality

The MEGA65 includes an F018/F018A backward-compatible DMA controller. Unlike in the C65, where the DMA controller exists as a separate chip, it is part of the 45GS02 processor in the MEGA65. However, as the use of the DMA controller is a logically separate topic, it is documented separately in this appendix.

The MEGA65's DMA controller provides several important improvements over the F018/F018A DMAgic chips of the C65:

- **Speed** The MEGA65 performs DMA operations at 40MHz, allowing filling 40MB or copying 20MB per second. For example, it is possible to copy a complete 8KB C64-style bitmap display in about 200 micro-seconds, equivalent to less than four raster lines!
- **Large Memory Access** The MEGA65's DMA controller allows access to all 256MB of address space.
- **Texture Copying Support** The MEGA65's DMA controller can do fractional address calculations to support hardware texture scaling, as well as address striding, to make it possible in principle to simultaneously scale-and-draw a texture from memory to the screen. This would be useful, should anyone be crazy enough to try to implement a Wolfenstein or Doom style-game on the MEGA65.
- **Transparency/Mask Value Support** The MEGA65's DMA controller can be told to ignore a special value when copying memory, leaving the destination memory contents unchanged. This allows masking of transparent regions when performing a DMA copy, which considerably simplifies blitting of graphics shapes.
- **Per-Job Option List** A number of options can be configured for each job in a chained list of DMA jobs, for example, selecting F018 or F018B mode, changing the transparency value, fractional address stepping or the source or destination memory region.
- **Background Audio DMA** The MEGA65 includes background audio DMA capabilities similar to the Amiga™ series of computers. Key differences are that the MEGA65 can use either 8 or 16-bit samples, supports very high sample rates up to approximately 1 MHz, has 256 volume settings per channel, and no inter-channel modulation.

F018A/B DMA JOBS

To execute a DMA job using the F018 series of DMA controllers, you must construct the list of DMA jobs in memory, and then write the address of this list into the DMA address registers. The DMA job will execute when you write to the ADDRLSBTRIG register ($D700). For this reason you must write the MSB and bank number of the DMA list int0 $D701 and $D702 first, and the LSB only after having set these other two registers. If you wish to execute multiple DMA jobs using the same list structure in memory, you can simply write to ADDRLSBTRIG again after updating the list contents – provided that no other program has modified the contents of $D701 or $D702. Note that BASIC 65

uses the DMA controller to scroll the screen, so it is usually safest to always write to all three registers.

When ADDRLSBTRIG has been written to, the DMA job completes immediately. Unlike on the C65, the DMA controller is part of the processor of the MEGA65. This means that the processor stops trying to execute instructions until the DMA job has completed. The only exception to this, is that Audio DMA continues, and will steal cycles from any other DMA activity, to ensure that audio playback is not affected.

This behaviour means that unlike on the C65, DMA jobs cannot be interrupted. If your program has sensitive timing requirements, you may need to break larger DMA jobs into several smaller jobs. This is somewhat mitigated by the high speed of the MEGA65's DMA, which is able to fill memory at 40.5MB per second and copy memory at 20.25MB per second, compared with circa 3.5MB and 1.7MB per second on a C65. This allows larger DMA jobs to be executed, without needing to worry about the impact on real-time elements of a program. For example, it is possible to fill an 80 column 50 row text screen using the MEGA65's DMA controller in just 200 microseconds.

F018 DMA Job List Format

The MEGA65's DMA controller supports the two different DMA job list formats used by the original F018 part that was used in the earlier C65 prototypes (upto Revision 2B) and the F018B and later revisions used in the Revision 3 - 5 C65 prototypes. The main difference is the addition of a second command byte, as the following tables show:

It is important to know which style the DMA controller is expecting. The MEGA65's Hypervisor sets the mode based on the detected version of C65 ROM, if one is running. If it is an older one, then the F018 style is expected, otherwise the newer F018B style is expected. You can check which style has been selected by querying bit 0 of $D703: If it is a 1, then the newer F018B 12 byte list format is expected. If it is a 0, then the older F018 11 byte list format is expected. The expected style can be set by writing to this register.

Unless you are writing software that must also run on a C65 prototype, you should most probably use the MEGA65's Enhanced DMA Jobs, where the list format is explicitly specified in the list itself. As the Enhanced DMA Jobs are an extension of the F018/F018B DMA jobs, you should still read the following, unless you are already familiar with the behaviour of the F018 DMA controller.

F018 11 byte DMA List Structure

Offset	Contents
$00	Command LSB
$01	Count LSB
$02	Count MSB
$03	Source Address LSB
$04	Source Address MSB
$05	Source Address BANK and FLAGS
$06	Destination Address LSB
$07	Destination Address MSB
$08	Destination Address BANK and FLAGS
$09	Modulo LSB
$0a	Modulo MSB

* The Command MSB is $00 when using this list format.

F018B 12 byte DMA List Structure

Offset	Contents
$00	Command LSB
$01	Count LSB
$02	Count MSB
$03	Source Address LSB
$04	Source Address MSB
$05	Source Address BANK and FLAGS
$06	Destination Address LSB
$07	Destination Address MSB
$08	Destination Address BANK and FLAGS
$09	Command MSB
$0a	Modulo LSB / Mode
$0b	Modulo MSB / Mode

The structure of the command word is as follows:

Bit(s)	Contents
0 - 1	DMA Operation Type
2	Chain (i.e., another DMA list follows)
3	Yield to interrupts
4	MINTERM -SA,-DA bit
5	MINTERM -SA,DA bit
6	MINTERM SA,-DA bit
7	MINTERM SA,DA bit
8 - 9	Addressing mode of source
10 - 11	Addressing mode of destination
12 - 15	RESESRVED. Always set to 0's

The command field take the following four values:

Value	Contents
%00 (0)	Copy
%01 (1)	Mix (via MINTERMs)
%10 (2)	Swap
%11 (3)	Fill

* Only Copy and Fill are implemented at the time of writing.

The addressing mode fields take the following four values:

Value	Contents
%00 (0)	Linear (normal) addressing
%01 (1)	Modulo (rectangular) addressing
%10 (2)	Hold (constant address)
%11 (3)	XY MOD (bitmap rectangular) addressing

* Only Linear, Modulo and Hold are implemented at the time of writing.

The BANK and FLAGS field for the source address allow selection of addresses within a 1MB address space. To access memory beyond the first 1MB, it is necessary to use an Enhanced DMA Job with the appropriate option bytes to select the source and/or destination MB of memory. The BANK and FLAGS field has the following structure:

Bit(s)	Contents
0 – 3	Memory BANK within the selected MB
4	HOLD, i.e., do not change the address
5	MODULO, i.e., apply the MODULO field to wrap-around within a limited memory space
6	DIRECTION. If set, then the address is decremented instead of incremented.
7	I/O. If set, then I/O registers are visible during the DMA controller at $D000 – $DFFF.

Performing Simple DMA Operations

For information on using the DMA controller from BASIC 65, refer to the **DMA** BASIC command in *the MEGA65 Book*, DMA (subsection B).

To use the DMA controller from assembly language, set up a data structure with the DMA list, and then set $D702 – $D700 to the address of the list. For example, to clear the screen in C65-mode by filling it with spaces, the following routine could be used:

```
  LDA #$00       ; DMA list exists in BANK 0
  STA $D702
  LDA #>dmalist  ; Set MSB of DMA list address
  STA $D701
  LDA #<dmalist  ; Set LSB of DMA list address, and execute DMA
  STA $D700
  RTS

dmalist:
  .byte $03   ; Command low byte: FILL
  .word 2000  ; Count: 80x25 = 2000 bytes
  .word $0020 ; Fill with value $20
  .byte $00   ; Source bank (ignored with FILL operation)
  .word $0800 ; Destination address where screen lives
  .byte $00   ; Screen is in bank 0
  .byte $00   ; Command high byte
  .word $0000 ; Modulo (ignored due to selected commmand)
```

It is also possible to execute more than one DMA job at the same time, by setting the CHAIN bit in the low byte of the command word. For example to clear the screen as above, and also clear the colour RAM for the screen, you could use something like:

```
  LDA #$00       ; DMA list exists in BANK 0
  STA $D702
  LDA #>dmalist  ; Set MSB of DMA list address
  STA $D701
  LDA #<dmalist  ; Set LSB of DMA list address, and execute DMA
  STA $D700
  RTS

dmalist:
  .byte $07   ; Command low byte: FILL + CHAIN
  .word 2000  ; Count: 80x25 = 2000 bytes
  .word $0020 ; Fill with value $20
  .byte $00   ; Source bank (ignored with FILL operation)
  .word $0800 ; Destination address where screen lives
  .byte $00   ; Screen is in bank 0
  .byte $00   ; Command high byte
  .word $0000 ; Modulo (ignored due to selected commmand)

  ; Second DMA job immediately follows the first
  .byte $03   ; Command low byte: FILL
  .word 2000  ; Count: 80x25 = 2000 bytes
  .word $0001 ; Fill with value $01 = white
  .byte $00   ; Source bank (ignored with FILL operation)
  .word $F800 ; Destination address where colour RAM lives
  .byte $01   ; colour RAM is in bank 1 ($1F800-$1FFFF)
  .byte $00   ; Command high byte
  .word $0000 ; Modulo (ignored due to selected commmand)
```

Copying memory is very similar to filling memory, except that the command low byte must be modified, and the source address field must be correctly initialised. For example, to copy the character set from where it lives in the ROM at $2D000 - $2DFFF to $5000, you could use something like:

```
  LDA #$00        ; DMA list exists in BANK 0
  STA $D702
  LDA #>dmalist   ; Set MSB of DMA list address
  STA $D701
  LDA #<dmalist   ; Set LSB of DMA list address, and execute DMA
  STA $D700
  RTS

dmalist:
  .byte $00   ; Command low byte: COPY
  .word $1000 ; Count: 4KB = 4096
  .word $D000 ; Copy from $xD000
  .byte $02   ; Source bank = $02 for $2xxxx
  .word $5000 ; Destination address where screen lives
  .byte $00   ; Screen is in bank 0
  .byte $00   ; Command high byte
  .word $0000 ; Modulo (ignored due to selected commmand)
```

It is also possible to perform a DMA operation from BASIC 2 in C64 mode by POKEing the necessary values, after first making sure that MEGA65 or C65 I/O mode has been selected by writing the appropriate values to $D02F (53295). For example, to clear the screen in C64 BASIC 2 using the DMA controller, you could use something like:

```
  10 rem enable mega65 I/O
  20 poke53295,asc("g"):poke53295,asc("s")
  30 rem dma list in data statements
  40 data 3: rem command lsb = fill
  50 data 232,3 : rem screen is 1000 bytes = 3*256+232
  60 data 32,0: rem fill with space = 32
  70 data 0: rem source bank (unused for fill)
  80 data 0,4: rem screen address = 1024 = 4*256
  90 data 0: rem screen lives in bank 0
  100 data 0: rem command high byte
  110 data 0,0: rem modulo (unused in this job)
  120 rem put dma list at $c000 = 49152
  130 fori=0to11:reada:poke49152+i,a:next
  140 rem execute job
  150 poke55042,0: rem dma list is in bank 0
  160 poke55041,192: rem dma list is in $c0xx
  170 poke55040,0: rem dma list is in $xx00, and execute
```

While this is rather cumbersome to do each time, if you wanted to clear the screen again, all you would need to do would be to **`POKE 55040,0`** again, assuming that the DMA list and DMA controller registers had not been modified since the previous time the DMA job had been run.

The HOLD, I/O and other options can also be used to create interesting effects. For example, to write a new value to the screen background colour very quickly, you could copy a region of memory to $D021, with the I/O flag set to make the I/O register visible for writing in the DMA job, and the HOLD flag set, so that the same address gets written to repeatedly. This will write to the background colour at a rate of 20.5MHz, which is almost as fast as the video pixel clock (27MHz). Thus we can change the colour almost every pixel.

With a little care, we can make this routine such that it takes exactly one raster-line to run, and thus draw vertical raster bars, or to create a kind of frankenstein video mode that uses a linear memory layout – at the cost of consuming all of the processor's time during the active part of the display.

The following example does this to draw vertical raster bars on the screen. This program assumes that the MEGA65 is set to PAL. For NTSC, the size of the DMA transfer would need to be decreased a little. The other thing to note with this program, is that it uses MEGA65 Enhanced DMA Job option $81 to set the destination megabyte in memory to $FFxxxxx, and the bank is set to $D, and the destination address to $0021, to form the complete address $FFD0021. This is the true location of the VIC-IV's border colour register. The program is written using ACME-compatible syntax.

```
basicheader:
††I;; 2020 SYS 2061
††I!word $80a,2020
††I!byte $9e,$32,$30,$36,$31,0,0,0

††I;; Actual code begining at $080d = 2061
main:
††Isei
††Ilda #$47        ††I; enable MEGA65 I/O
††Ista $D02f
††Ilda #$53
††Ista $d02f
††Ilda #65 ††I††I; Set CPU speed to fast
††Ista 0
††Ilda #0            ††I; disable screen to show only the border
††Ista $d011

††Ilda $d012              ††I; Wait until start of the next raster
raster+sync:††I  ††I††I; before beginning loop for horizontal alignment
††Icmp $d012
††Ibeq raster+sync

††I;; The following loop takes exactly one raster line at 40.5MHz in PAL
loop:
††Ijsr triggerdma
††Ijmp loop

triggerdma:
††Ilda #0††I††I††I; make sure F018 list format
††Ista $d703

††Ilda #0     ††I††I; dma list bank
††Ista $d702
††Ilda #>rasterdmalist
††Ista $d701
††Ilda #<rasterdmalist
††Ista $d705
††Irts
```

```
rasterdmalist:
↑↑I!byte $81,$ff,$00
↑↑I!byte $00 ↑↑I↑↑I; COPY
↑↑I!word 619 ↑↑I↑↑I; DMA transfer is 619 bytes long
↑↑I!word rastercolours↑↑I; source address
↑↑I!byte $00               ; source bank
↑↑I!word $0020↑↑I↑↑I; destination address
↑↑I!byte $1d               ; destination bank + HOLD
↑↑I;; unused modulo field
↑↑I!word $0000
```

```
rastercolours:
↑↑I!byte 0,0,0,0,0,0,0,0,0,0,0,0,0,0
↑↑I!byte 0,0,0,11,11,11,12,12,12,15,15,15,1,1,1,15,15,15,12,12,12,11,11,11,0,0,0
↑↑I!byte 0,0,0,6,6,6,4,4,4,14,14,14,3,3,3,1,1,1,3,3,3,14,14,14,4,4,4,6,6,6,0,0,0
↑↑I!byte 0,0,0,11,11,11,12,12,12,15,15,15,1,1,1,15,15,15,12,12,12,11,11,11,0,0,0
↑↑I!byte 0,0,0,6,6,6,4,4,4,14,14,14,3,3,3,1,1,1,3,3,3,14,14,14,4,4,4,6,6,6,0,0,0
↑↑I!byte 0,0,0,11,11,11,12,12,12,15,15,15,1,1,1,15,15,15,12,12,12,11,11,11,0,0,0
↑↑I!byte 0,0,0,6,6,6,4,4,4,14,14,14,3,3,3,1,1,1,3,3,3,14,14,14,4,4,4,6,6,6,0,0,0
↑↑I!byte 0,0,0,11,11,11,12,12,12,15,15,15,1,1,1,15,15,15,12,12,12,11,11,11,0,0,0
↑↑I!byte 0,0,0,6,6,6,4,4,4,14,14,14,3,3,3,1,1,1,3,3,3,14,14,14,4,4,4,6,6,6,0,0,0
↑↑I!byte 0,0,0,11,11,11,12,12,12,15,15,15,1,1,1,15,15,15,12,12,12,11,11,11,0,0,0
↑↑I!byte 0,0,0,6,6,6,4,4,4,14,14,14,3,3,3,1,1,1,3,3,3,14,14,14,4,4,4,6,6,6,0,0,0
↑↑I!byte 0,0,0,11,11,11,12,12,12,15,15,15,1,1,1,15,15,15,12,12,12,11,11,11,0,0,0
↑↑I!byte 0,0,0,6,6,6,4,4,4,14,14,14,3,3,3,1,1,1,3,3,3,14,14,14,4,4,4,6,6,6,0,0,0
↑↑I!byte 0,0,0,11,11,11,12,12,12,15,15,15,1,1,1,15,15,15,12,12,12,11,11,11,0,0,0
↑↑I!byte 0,0,0,6,6,6,4,4,4,14,14,14,3,3,3,1,1,1,3,3,3,14,14,14,4,4,4,6,6,6,0,0,0
↑↑I!byte 0,0,0,11,11,11,12,12,12,15,15,15,1,1,1,15,15,15,12,12,12,11,11,11,0,0,0
↑↑I!byte 0,0,0,6,6,6,4,4,4,14,14,14,3,3,3,1,1,1,3,3,3,14,14,14,4,4,4,6,6,6,0,0,0
↑↑I!byte 0,0,0,11,11,11,12,12,12,15,15,15,1,1,1,15,15,15,12,12,12,11,11,11,0,0,0
↑↑I!byte 0,0,0,6,6,6,4,4,4,14,14,14,3,3,3,1,1,1,3,3,3,14,14,14,4,4,4,6,6,6,0,0,0
```

MEGA65 ENHANCED DMA JOBS

The MEGA65's implementation of the DMAgic supports significantly enhanced DMA jobs. An enhanced DMA job is indicated by writing the low byte of the DMA list address to $D705 instead of to $D700. The MEGA65 will then look for one or more *job option tokens* at the start of the DMA list. Those tokens will be interpretted, before executing the DMA job which immediately follows the *end of job options* token ($00).

Job option tokens that take an argument have the most-significant bit set, and always take a 1 byte option. Job option tokens that take no argument have the most-significant-bit clear. Unsupported job option tokens are simply ignored. This allows for

future revisions of the DMAgic to add support for additional options, without breaking backward compatibility.

These options are also used to achieve advanced features, such as hardware texture scaling at up to 20Mpixels per second, and hardware line drawing at up to 40Mpixels per second. These advanced functions are implemented by allowing complex calculations to be made to the source and/or destination address of DMA jobs as they execute.

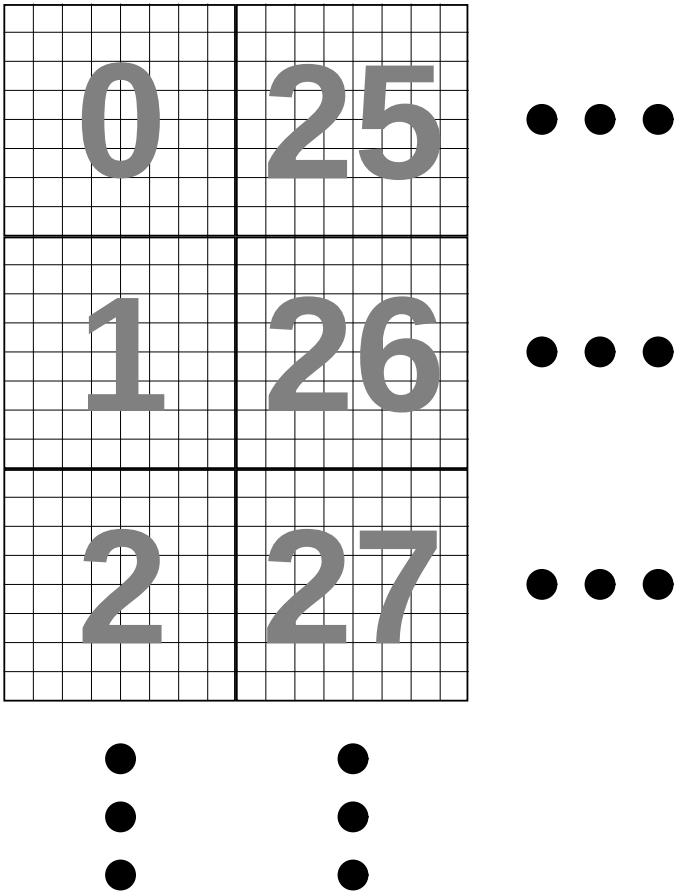
The list of valid job option tokens is:

$00	End of job option list
$06	Disable use of transparent value
$07	Enable use of transparent value
$0A	Use 11 byte F011A DMA list format
$0B	Use 12 byte F011B DMA list format
$0D	Write raw flux to floppy drive (see **??**)
$0E	Read raw flux to floppy drive (see **??**)
$0F	Read raw flux to floppy drive (see **??**)
$53	Enable 'Shallan Spiral' Mode
$80	Source address bits 20 - 27
$81	Destination address bits 20 - 27
$82	Source skip rate (256^{ths} of bytes)
$83	Source skip rate (whole bytes)
$84	Destination skip rate (256^{ths} of bytes)
$85	Destination skip rate (whole bytes)
$86	Transparent value (bytes with matching value are not written)
$87	Set X column bytes (LSB) for line drawing destination address
$88	Set X column bytes (MSB) for line drawing destination address
$89	Set Y row bytes (LSB) for line drawing destination address
$8A	Set Y row bytes (MSB) for line drawing destination address
$8B	Slope (LSB) for line drawing destination address
$8C	Slope (MSB) for line drawing destination address
$8D	Slope accumulator initial fraction (LSB) for line drawing destination address
$8E	Slope accumulator initial fraction (MSB) for line drawing destination address
$8F	Line Drawing Mode enable and options for destination address (set in argument byte): Bit 7 = enable line mode, Bit 6 = select X or Y direction, Bit 5 = slope is negative.
$97	Set X column bytes (LSB) for line drawing source address
$98	Set X column bytes (MSB) for line drawing source address
$99	Set Y row bytes (LSB) for line drawing source address
$9A	Set Y row bytes (MSB) for line drawing source address
$9B	Slope (LSB) for line drawing source address
$9C	Slope (MSB) for line drawing source address
$9D	Slope accumulator initial fraction (LSB) for line drawing source address
$9E	Slope accumulator initial fraction (MSB) for line drawing source address
$9F	Line Drawing Mode enable and options for source address (set in argument byte): Bit 7 = enable line mode, Bit 6 = select X or Y direction, Bit 5 = slope is negative.

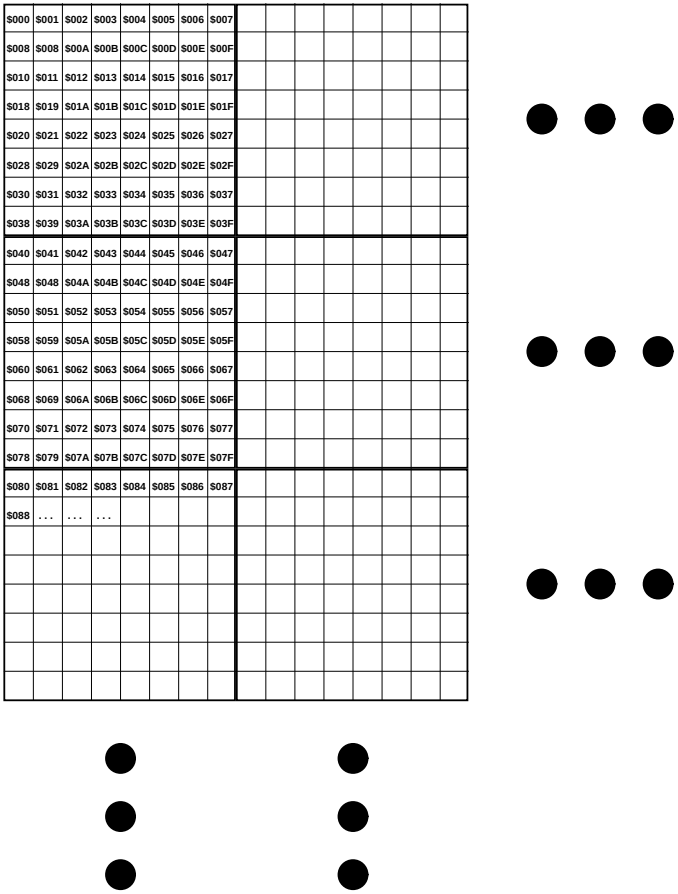
TEXTURE SCALING AND LINE DRAWING

The DMAgic supports an advanced internal address calculator that allows it to draw scaled textures and draw lines with arbitrary slopes on VIC-IV FCM video displays.

For texture scaling, the FCM screen must be arranged vertically, as shown below:



By lining the characters into vertical columns like this, advancing vertically by one pixel adds a constant 8 bytes each time, as shown below:



The source and destination skip rates also allow setting the scaling factors. A skip rate of $0100 this corresponds to stepping $01.00 pixels. To use the vertically stacked FCM layout as the target for copying vertical lines of textrures, then the destination skip rate should be $0800, i.e., 8.0 bytes per pixel. This would copy a vertical line of texture data without scaling. By setting the source stepping to < $0100 will cause some pixels to be repeated, effectively zooming the texture in, while setting the source stepping to > $0100 will cause some pixels to be skipped, effectively zooming the texture out. The destination stepping does not ordinary need to be adjusted. Note that the texture data must be stored with each vertical stripe stored contiguously, so that this mode can be used.

For line drawing, the DMA controller needs to know the screen layout, specifically, what number must be added to the address of a rightmost pixel in one column of FCM characters in order to calculate the address of the pixel appearing immediately to its right. Similarly, it must also know how much must be added to the address of a bottom most pixel in one row of FCM characters in order to calculate the address of the pixel

appearing immediately below it. This allows for flexible screen layout options, and arbitrary screen sizes. You must then also specify the slope of the line, and whether the line has the X or Y as its major axis, and whether the slope is positive or negative.

The file `test_290.c` in the https://github.com/mega65/mega65-tools repository provides an example of using these facilities to implement hardware accelerated line drawing. This is *very* fast, as it draws lines at the full DMA fill speed, i.e., approximately 40,500,000 pixels per second.

INLINE DMA LISTS

Normally you have to setup a separate area of memory that contains the DMA list, and then load the address of that area into the DMA address registers at $D70x. Because the MEGA65's DMA controller is part of the CPU, it supports an additional mode that is very convenient, called inline DMA list mode.

This mode works like Enhanced DMA mode, except that the DMA list is read starting from the current value of the Program Counter (PC) register. To use this mode, write any value to $D707, to immediately a trigger a DMA job, with the list in the bytes immediately following the instruction that writes to $D707.

The DMA list can be a single job, or chained, as with any other DMA job. The real magic is that the Program Counter gets set to the next address after the end of the DMA list, and that the DMA list is read from the CPU's current memory mapping. This means that you can execute code with DMA lists from any bank of memory, without having to worry about which bank it is in.

For example, the following code would clear the C65-mode screen, before flashing the border endlessly:

```
        STA $D707
        .byte $00   ; end of job options
        .byte $03   ; fill
        .word 2000  ; count
        .word $0020 ; value
        .byte $00   ; src bank
        .word $0800 ; dst
        .byte $00   ; dst bank
        .byte $00   ; cmd hi
        .word $0000 ; modulo / ignored
foo:
        INC $d020
        JMP foo
```

AUDIO DMA

The MEGA65 includes four channels of DMA-driven audio playback that can be used in place of the direct digital audio registers at $D6F8-$D6FB. That is, you must select which of these two sources to feed to the audio cross-bar mixer. This is selected via the AUDEN signal ($D711 bit 7), which simultaneously enables the audio DMA function in the processor, as well as instructing the audio cross-bar mixer to use the audio from this instead of the $D6F8-$D6FB digital audio registers. If you wish to have no other audio than the audio DMA channels, the audio cross-bar mixer can be bypassed, and the DMA audio played at full volume by setting the NOMIX signal ($D711 bit 4). In that mode no audio from the SIDs, FM, microphones or other sources will be available. All other bits in $D711 should ordinarily be left clear, i.e., write $80 to $D711 to enable audio DMA.

Two channels form the left digital audio channel, and the other two channels form the right digital audio channel. It is these left and right channels that are then fed into the MEGA65's audio cross-bar mixer.

As the DMA controller is part of the processor of the MEGA65, and the MEGA65 does not have reserved bus slots for multi-media operations, the MEGA65 uses idle CPU cycles to perform background DMA. This requires that the MEGA65 CPU be set to the "full speed" mode, i.e., approximately 40MHz. In this mode, there is a wait-state whenever reading an operand from memory. Thus each instruction that loads a byte from memory will create one implicit audio DMA slot. This is rarely a problem in practice, except if the processor idles in a very tight loop. To ensure that audio continues to play in the background, such loops should include a read instruction, such as:

```
loop:   LDA $1234   // Ensure loop has at least one idle cycle for
                    // audio DMA
        JMP loop
```

Each of the four DMA channels is configured using a block of 16 registers at $D720, $D730, $D740 and $D750, respectively. We will explain the registers for the first channel, channel 0, at $D720 - $D72F.

Sample Address Management

To play an audio sample you must first supply the start address of the sample. This is a 24-bit address, and must be in the main chip memory of the MEGA65. This is done by writing the address into $D72A - $D72C. This is the address of the first sample value that will be played. You must then provide the end address of the sample in $D727 - $D728. But note that this is is only 16 bits. This is because the MEGA65 compares only the bottom 16 bits of the address when checking if it has reached the end of a sample. In practice, this means that samples cannot be more than 64KB in size. If the sample contains a section that should be repeated, then the start address of the

repeating part should be loaded into $D721 - $D723, and the CH0LOOP bit should be set ($D720 bit 6).

You can determine the current sample address at any time by reading the registers at $D72A - $D72C. But beware: These registers are not latched, so it is possible that the values may be updated as you read the registers, unless you stop the channel first by clearing the CH0EN signal.

Sample Playback frequency and Volume

The MEGA65 controls the playback rate of audio DMA samples by using a 24-bit counter. Whenever the 24-bit counter overflows, the next sample value is requested. Sample speed control is achieved by setting the value added to this counter each CPU cycle. Thus a value of $FFFFFF would result in a sample rate of almost 40.5 MHz. In practice, sample rates above a few megahertz are not possible, because there are insufficient idle CPU cycles, and distorted audio will result. Even below this, care must be taken to ensure that idle cycles come sufficiently often and dispersed throughout the processor's instruction stream to prevent distortion. At typical sample rates below 16KHz and using 8-bit samples these effects are typically negligible for normal instruction streams, and so no special action is normally required for typical audio playback.

At the other end of the scale, sample rates as low as $40.5\text{MHz}/2^{24}$ = 2.4 samples per second are possible. This is sufficiently low enough for even the most demanding infra-sound applications.

Volume is controlled by setting $D729. Maximum volume is obtained with the value $FF, while a value of $00 will effectively mute the channel. The first two audio channels are normally allocated to the left, and the second two to the right. However, the MEGA65 includes separate volume controls for the opposite channels. For example, to play audio DMA channel 0 at full volume on both left and right-hand sides of the audio output, set both $D729 and $D71C to $FF. This allows panning of the four audio DMA channels.

Both the frequency and volume can be freely adjusted while a sample is playing to produce various effects.

Pure Sine Wave

Where it is necessary to produce a stable sine wave, especially at higher frequencies, there is a special mode to support this. By setting the CH0SINE signal, the audio channel will play a 32 byte 16-bit sine wave pattern. The sample addresses still need to be set, as though the sine wave table were located in the bottom 64 bytes of memory, as the normal address generation logic is used in this mode. However, no audio DMA fetches are performed when a channel is in this mode, thus avoiding all sources of distortion due to irregular spacing of idle cycles in the processor's instruction stream.

This can be used to produce sine waves in both the audible range, as well as well into the ultrasonic range, at frequencies exceeding 60,000Hz, provided that the MEGA65 is connected to an appropriately speaker arrangement.

Sample playback control

To begin a channel playing a sample, set the CH0EN signal ($D720 bit 7). The sample will play until its completion, unless the CH0LOOP signal has also been set. When a sample completes playing, the CH0STP flag will be set. The audio DMA subsystem cannot presently generate interrupts.

Unlike on the Amiga™, the MEGA65 audio DMA system supports both 8 and 16-bit samples. It also supports packed 4-bit samples, playing either the lower or upper nibble of each sample byte. This allows two separate samples to occupy the same byte, thus effectively halving the amount of space required to store two equal length samples.

F018 "DMAGIC" DMA CONTROLLER

HEX	DEC	Signal	Description
D700	55040	ADDRLSB-TRIG	DMAgic DMA list address LSB, and trigger DMA (when written)
D701	55041	ADDRMSB	DMA list address high byte (address bits 8 - 15).
D702	55042	ADDRBANK	DMA list address bank (address bits 16 - 22). Writing clears $D704.

MEGA65 DMA CONTROLLER EXTENSIONS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D703	55043	-							EN018B
D704	55044	ADDRMB							
D705	55045	ETRIG							
D706	55046	ETRIGMAPD							
D70E	55054	ADDRLSB							
D711	55057	AUDEN	BLKD	AUD-WRBLK	NOMIX	-	AUDBLKTO		
D71C	55068	CH0RVOL							
D71D	55069	CH1RVOL							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D71E	55070	CH2LVOL							
D71F	55071	CH3LVOL							
D720	55072	CH0EN	CH0LOOP	CH0SGN	CH0SINE	CH0STP	–	CH0SBITS	
D721	55073	CH0BADDRL							
D722	55074	CH0BADDRC							
D723	55075	CH0BADDRM							
D724	55076	CH0FREQL							
D725	55077	CH0FREQC							
D726	55078	CH0FREQM							
D727	55079	CH0TADDRL							
D728	55080	CH0TADDRM							
D729	55081	CH0VOLUME							
D72A	55082	CH0CURADDRL							
D72B	55083	CH0CURADDRC							
D72C	55084	CH0CURADDRM							
D72D	55085	CH0TMRADDRL							
D72E	55086	CH0TMRADDRC							
D72F	55087	CH0TMRADDRM							
D730	55088	CH1EN	CH1LOOP	CH1SGN	CH1SINE	CH1STP	–	CH1SBITS	
D731	55089	CH1BADDRL							
D732	55090	CH1BADDRC							
D733	55091	CH1BADDRM							
D734	55092	CH1FREQL							
D735	55093	CH1FREQC							
D736	55094	CH1FREQM							
D737	55095	CH1TADDRL							
D738	55096	CH1TADDRM							
D739	55097	CH1VOLUME							
D73A	55098	CH1CURADDRL							
D73B	55099	CH1CURADDRC							
D73C	55100	CH1CURADDRM							
D73D	55101	CH1TMRADDRL							
D73E	55102	CH1TMRADDRC							
D73F	55103	CH1TMRADDRM							
D740	55104	CH2EN	CH2LOOP	CH2SGN	CH2SINE	CH2STP	–	CH2SBITS	
D741	55105	CH2BADDRL							
D742	55106	CH2BADDRC							
D743	55107	CH2BADDRM							
D744	55108	CH2FREQL							
D745	55109	CH2FREQC							
D746	55110	CH2FREQM							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D747	55111	CH2TADDRL							
D748	55112	CH2TADDRM							
D749	55113	CH2VOLUME							
D74A	55114	CH2CURADDRL							
D74B	55115	CH2CURADDRC							
D74C	55116	CH2CURADDRM							
D74D	55117	CH2TMRADDRL							
D74E	55118	CH2TMRADDRC							
D74F	55119	CH2TMRADDRM							
D750	55120	CH3EN	CH3LOOP	CH3SGN	CH3SINE	CH3STP	-	CH3SBITS	
D751	55121	CH3BADDRL							
D752	55122	CH3BADDRC							
D753	55123	CH3BADDRM							
D754	55124	CH3FREQL							
D755	55125	CH3FREQC							
D756	55126	CH3FREQM							
D757	55127	CH3TADDRL							
D758	55128	CH3TADDRM							
D759	55129	CH3VOLUME							
D75A	55130	CH3CURADDRL							
D75B	55131	CH3CURADDRC							
D75C	55132	CH3CURADDRM							
D75D	55133	CH3TMRADDRL							
D75E	55134	CH3TMRADDRC							
D75F	55135	CH3TMRADDRM							

- **ADDRLSB** DMA list address low byte (address bits 0 - 7) WITHOUT STARTING A DMA JOB (used by Hypervisor for unfreezing DMA-using tasks)
- **ADDRMB** DMA list address mega-byte
- **AUDBLKTO** Audio DMA block timeout (read only) DEBUG
- **AUDEN** Enable Audio DMA
- **AUDWRBLK** Audio DMA block writes (samples still get read)
- **BLKD** Audio DMA blocked (read only) DEBUG
- **CH0RVOL** Audio DMA channel 0 right channel volume
- **CH1RVOL** Audio DMA channel 1 right channel volume
- **CH2LVOL** Audio DMA channel 2 left channel volume
- **CH3LVOL** Audio DMA channel 3 left channel volume

- **CHXBADDRC** Audio DMA channel X base address middle byte
- **CHXBADDRL** Audio DMA channel X base address LSB
- **CHXBADDRM** Audio DMA channel X base address MSB
- **CHXCURADDRC** Audio DMA channel X current address middle byte
- **CHXCURADDRL** Audio DMA channel X current address LSB
- **CHXCURADDRM** Audio DMA channel X current address MSB
- **CHXEN** Enable Audio DMA channel X
- **CHXFREQC** Audio DMA channel X frequency middle byte
- **CHXFREQL** Audio DMA channel X frequency LSB
- **CHXFREQM** Audio DMA channel X frequency MSB
- **CHXLOOP** Enable Audio DMA channel X looping
- **CHXSBITS** Audio DMA channel X sample bits (11=16, 10=8, 01=upper nybl, 00=lower nybl)
- **CHXSGN** Enable Audio DMA channel X signed samples
- **CHXSINE** Audio DMA channel X play 32-sample sine wave instead of DMA data
- **CHXSTP** Audio DMA channel X stop flag
- **CHXTADDRL** Audio DMA channel X top address LSB
- **CHXTADDRM** Audio DMA channel X top address MSB
- **CHXTMRADDRC** Audio DMA channel X timing counter middle byte
- **CHXTMRADDRL** Audio DMA channel X timing counter LSB
- **CHXTMRADDRM** Audio DMA channel X timing counter MSB
- **CHXVOLUME** Audio DMA channel X playback volume
- **EN018B** DMA enable F018B mode (adds sub-command byte)
- **ETRIG** Set low-order byte of DMA list address, and trigger Enhanced DMA job, with list address specified as 28-bit flat address (uses DMA option list)
- **ETRIGMAPD** Set low-order byte of DMA list address, and trigger Enhanced DMA job, with list in current CPU memory map (uses DMA option list)
- **NOMIX** Audio DMA bypasses audio mixer

UNIMPLEMENTED FUNCTIONALITY

The MEGA65's DMAgic does not currently support either memory-swap or mini-term operations.

Miniterms were intended for bitplane blitting, which is not required for the MEGA65 which offers greatly advanced character modes and stepped and fractional DMA address incrementing which allows efficient texture copying and scaling. Also there exists no known software which ever used this facility, and it remains uncertain if it was ever implemented in any revision of the DMAgic chip used in C65 prototypes.

The memory-swap operation is intended to be implemented, but can be worked around in the meantime by copying the first region to a 3rd region that acts as a temporary buffer, then copying the 2nd region to the 1st, and the 3rd to the 2nd.

CHAPTER 5

6526 Complex Interface Adapter (CIA) Registers

- CIA 6526 Registers
- CIA 6526 Hypervisor Registers

CIA 6526 REGISTERS

CIA 1 Registers

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DC00	56320	PORTA							
DC01	56321	PORTB							
DC02	56322	DDRA							
DC03	56323	DDRB							
DC04	56324	TIMERA							
DC05	56325	TIMERA							
DC06	56326	TIMERB							
DC07	56327	TIMERB							
DC08	56328	-				TODJIF			
DC09	56329	-		TODSEC					
DC0A	56330	-		TODMIN					
DC0B	56331	TOD-AMPM	-		TODHOUR				
DC0C	56332	SDR							
DC0D	56333	IR	ISRCLR		FLG	SP	ALRM	TB	TA
DC0E	56334	TOD50	SPMOD	IMODA	-	RMODA	OMODA	PBONA	STRTA
DC0F	56335	TODEDIT	IMODB		LOAD	RMODB	OMODB	PBONB	STRTB

- **ALRM** TOD alarm
- **DDRA** Port A DDR
- **DDRB** Port B DDR
- **FLG** FLAG edge detected
- **IMODA** Timer A tick source
- **IMODB** Timer B tick source
- **IR** Interrupt flag
- **ISRCLR** Placeholder - Reading clears events
- **LOAD** Strobe input to force-load timers
- **OMODA** Timer A toggle or pulse
- **OMODB** Timer B toggle or pulse
- **PBONA** Timer A PB6 out
- **PBONB** Timer B PB7 out
- **PORTA** Port A

- **PORTB** Port B
- **RMODA** Timer A one-shot mode
- **RMODB** Timer B one-shot mode
- **SDR** shift register data register(writing starts sending)
- **SP** shift register full/empty
- **SPMOD** Serial port direction
- **STRTA** Timer A start
- **STRTB** Timer B start
- **TA** Timer A underflow
- **TB** Timer B underflow
- **TIMERA** Timer A counter (16 bit)
- **TIMERB** Timer B counter (16 bit)
- **TOD50** 50/60Hz select for TOD clock
- **TODAMPM** TOD PM flag
- **TODEDIT** TOD alarm edit
- **TODHOUR** TOD hours
- **TODJIF** TOD tenths of seconds
- **TODMIN** TOD minutes
- **TODSEC** TOD seconds

CIA2 Registers

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DD00	56576	PORTA							
DD01	56577	PORTB							
DD02	56578	DDRA							
DD03	56579	DDRB							
DD04	56580	TIMERA							
DD05	56581	TIMERA							
DD06	56582	TIMERB							
DD07	56583	TIMERB							
DD08	56584	-				TODJIF			
DD09	56585	-		TODSEC					
DD0B	56587	TOD-AMPM	-		TODHOUR				

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DD0C	56588	SDR							
DD0D	56589	IR	ISRCLR		FLG	SP	ALRM	TB	TA
DD0E	56590	TOD50	SPMOD	IMODA	–	RMODA	OMODA	PBONA	STRTA
DD0F	56591	TODEDIT	IMODB		LOAD	RMODB	OMODB	PBONB	STRTB

- **ALRM** TOD alarm
- **DDRA** Port A DDR
- **DDRB** Port B DDR
- **FLG** FLAG edge detected
- **IMODA** Timer A tick source
- **IMODB** Timer B tick source
- **IR** Interrupt flag
- **ISRCLR** Placeholder - Reading clears events
- **LOAD** Strobe input to force-load timers
- **OMODA** Timer A toggle or pulse
- **OMODB** Timer B toggle or pulse
- **PBONA** Timer A PB6 out
- **PBONB** Timer B PB7 out
- **PORTA** Port A
- **PORTB** Port B
- **RMODA** Timer A one-shot mode
- **RMODB** Timer B one-shot mode
- **SDR** shift register data register(writing starts sending)
- **SP** shift register full/empty
- **SPMOD** Serial port direction
- **STRTA** Timer A start
- **STRTB** Timer B start
- **TA** Timer A underflow
- **TB** Timer B underflow
- **TIMERA** Timer A counter (16 bit)

- **TIMERB** Timer B counter (16 bit)
- **TOD50** 50/60Hz select for TOD clock
- **TODAMPM** TOD PM flag
- **TODEDIT** TOD alarm edit
- **TODHOUR** TOD hours
- **TODJIF** TOD tenths of seconds
- **TODSEC** TOD seconds

CIA 6526 HYPERVISOR REGISTERS

In addition to the standard CIA registers available on the C64 and C65, the MEGA65 provides an additional set of registers that are visible only when the system is in Hypervisor Mode. These additional registers allow the internal state of the CIA to be more fully extracted when freezing, thus allowing more programs to function correctly after being frozen. They are not visible when using the MEGA65 normally, and can be safely ignored by programmers who are not programming the MEGA65 in Hypervisor Mode.

CIA1 Hypervisor Registers

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DC10	56336	TALATCH							
DC11	56337	TALATCH							
DC12	56338	TALATCH							
DC13	56339	TALATCH							
DC14	56340	TALATCH							
DC15	56341	TALATCH							
DC16	56342	TALATCH							
DC17	56343	TALATCH							
DC18	56344	IMFLG	IMSP	IMALRM	IMTB	TODJIF			
DC19	56345	TODSEC							
DC1A	56346	TODMIN							
DC1B	56347	TOD-AMPM	TODHOUR						
DC1C	56348	DD00-DELAY	ALRMJIF						
DC1D	56349	ALRMSEC							
DC1E	56350	ALRMMIN							
DC1F	56351	ALRM-AMPM	ALRMHOUR						

- **ALRMAMPM** TOD Alarm AM/PM flag
- **ALRMHOUR** TOD Alarm hours value
- **ALRMJIF** TOD Alarm 10ths of seconds value (actually all 8 bits)
- **ALRMMIN** TOD Alarm minutes value
- **ALRMSEC** TOD Alarm seconds value
- **DD00DELAY** Enable delaying writes to $DD00 by 3 cycles to match real 6502 timing
- **IMALRM** Interrupt mask for TOD alarm
- **IMFLG** Interrupt mask for FLAG line
- **IMSP** Interrupt mask for shift register (serial port)
- **IMTB** Interrupt mask for Timer B
- **TALATCH** Timer A latch value (16 bit)
- **TODAMPM** TOD AM/PM flag
- **TODHOUR** TOD hours value
- **TODJIF** TOD 10ths of seconds value
- **TODMIN** TOD Alarm minutes value
- **TODSEC** TOD Alarm seconds value

CIA2 Hypervisor Registers

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DD10	56592	TALATCH							
DD11	56593	TALATCH							
DD12	56594	TALATCH							
DD13	56595	TALATCH							
DD14	56596	TALATCH							
DD15	56597	TALATCH							
DD16	56598	TALATCH							
DD17	56599	TALATCH							
DD18	56600	IMFLG	IMSP	IMALRM	IMTB	TODJIF			
DD19	56601	TODSEC							
DD1A	56602	TODMIN							
DD1B	56603	TOD-AMPM	TODHOUR						
DD1C	56604	DD00-DELAY	ALRMJIF						
DD1D	56605	ALRMSEC							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DD1E	56606	ALRMMIN							
DD1F	56607	ALRM-AMPM	ALRMHOUR						

- **ALRMAMPM** TOD Alarm AM/PM flag
- **ALRMHOUR** TOD Alarm hours value
- **ALRMJIF** TOD Alarm 10ths of seconds value (actually all 8 bits)
- **ALRMMIN** TOD Alarm minutes value
- **ALRMSEC** TOD Alarm seconds value
- **DD00DELAY** Enable delaying writes to $DD00 by 3 cycles to match real 6502 timing
- **IMALRM** Interrupt mask for TOD alarm
- **IMFLG** Interrupt mask for FLAG line
- **IMSP** Interrupt mask for shift register (serial port)
- **IMTB** Interrupt mask for Timer B
- **TALATCH** Timer A latch value (16 bit)
- **TODAMPM** TOD AM/PM flag
- **TODHOUR** TOD hours value
- **TODJIF** TOD 10ths of seconds value
- **TODMIN** TOD Alarm minutes value
- **TODSEC** TOD Alarm seconds value

CHAPTER 6

4551 UART, GPIO and Utility Controller

- C65 6551 UART Registers
- 4551 General Purpose I/O & Miscellaneous Interface Registers

C65 6551 UART REGISTERS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D600	54784	DATA							
D601	54785	-				FRMERR	PTYERR	RXOVR-RUN	RXRDY
D602	54786	TXEN	RXEN	SYNCMOD		CHARSZ		PTYEN	PTYEVEN
D603	54787	DIVISOR							
D604	54788	DIVISOR							
D605	54789	IMTXIRQ	IMRXIRQ	IMTXNMI	IMRXNMI	-			
D606	54790	IFTXIRQ	IFRXIRQ	IFTXNMI	IFRXNMI	-			

- **CHARSZ** UART character size: 00=8, 01=7, 10=6, 11=5 bits per byte
- **DATA** UART data register (read or write)
- **DIVISOR** UART baud rate divisor (16 bit). Baud rate = 7.09375MHz / DIVISOR, unless MEGA65 fast UART mode is enabled, in which case baud rate = 80MHz / DIVISOR
- **FRMERR** UART RX framing error flag (clear by reading $D600)
- **IFRXIRQ** UART interrupt flag: IRQ on RX (not yet implemented on the MEGA65)
- **IFRXNMI** UART interrupt flag: NMI on RX (not yet implemented on the MEGA65)
- **IFTXIRQ** UART interrupt flag: IRQ on TX (not yet implemented on the MEGA65)
- **IFTXNMI** UART interrupt flag: NMI on TX (not yet implemented on the MEGA65)
- **IMRXIRQ** UART interrupt mask: IRQ on RX (not yet implemented on the MEGA65)
- **IMRXNMI** UART interrupt mask: NMI on RX (not yet implemented on the MEGA65)
- **IMTXIRQ** UART interrupt mask: IRQ on TX (not yet implemented on the MEGA65)
- **IMTXNMI** UART interrupt mask: NMI on TX (not yet implemented on the MEGA65)
- **PTYEN** UART Parity enable: 1=enabled
- **PTYERR** UART RX parity error flag (clear by reading $D600)
- **PTYEVEN** UART Parity: 1=even, 0=odd
- **RXEN** UART enable receive
- **RXOVRRUN** UART RX overrun flag (clear by reading $D600)
- **RXRDY** UART RX byte ready flag (clear by reading $D600)
- **SYNCMOD** UART synchronisation mode flags (00=RX & TX both async, 01=RX sync, TX async, 1x=TX sync, RX async (unused on the MEGA65)

- **TXEN** UART enable transmit

4551 GENERAL PURPOSE I/O & MISCELLANEOUS INTERFACE REGISTERS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D609	54793	-							UFAST
D60A	54794	KEYQUEUE	MODKEY-CAPS	MOD-KEYSCRL	MOD-KEYALT	MOD-KEYMEGA	MODK-EYCTRL	MOD-KEYRSHFT	MOD-KEYLSHFT
D60B	54795	OSKZEN	OSKZON	PORTF					
D60C	54796	PORTFDDR		PORTFDDR					
D60D	54797	HDSCL	HDSDA	SDBSH	SDCS	SDCLK	SDDATA	RST41	CONN41
D60E	54798	BASHDDR							
D60F	54799	AC-CESSKEY	OSKDIM	REALHW	-			KEYUP	KEYLEFT
D610	54800	ASCIIKEY							
D611	54801	MDISABLE	MCAPS	MSCRL	MALT	MMEGA	MCTRL	MRSHFT	MLSHFT
D612	54802	LJOYB	LJOYA	JOYSWAP	OSKDE-BUG	-			
D615	54805	OSKEN	VIRTKEY1						
D616	54806	OSKALT	VIRTKEY2						
D617	54807	OSKTOP	VIRTKEY3						
D618	54808	KSCNRATE							
D619	54809	PETSCIIKEY							
D61A	54810	SYSCTL							
D61D	54813	KEYLED-ENA	KEYLEDREG						
D61E	54814	KEYLEDVAL							
D620	54816	POTAX							
D621	54817	POTAY							
D622	54818	POTBX							
D623	54819	POTBY							
D625	54821	J21L							
D626	54822	J21H							
D627	54823	J21LDDR							
D628	54824	BOARDMINOR				J21HDDR			
D629	54825	M65MODEL							

- **ACCESSKEY** Enable accessible keyboard input via joystick port 2 fire button

- **ASCIIKEY** Top of typing event queue as ASCII. Write to clear event ready for next.
- **BASHDDR** Data Direction Register (DDR) for $D60D bit bashing port.
- **BOARDMINOR** Read PCB minor revision (R5+ only, else reads zeroes)
- **CONN41** Internal 1541 drive connect (1=connect internal 1541 drive to IEC bus)
- **HDSCL** HDMI I2C control interface SCL clock
- **HDSDA** HDMI I2C control interface SDA data line
- **J21H** J21 pins 11 - 14 input/output values
- **J21HDDR** J21 pins 11 - 14 data direction register
- **J21L** J21 pins 1 - 6, 9 - 10 input/output values
- **J21LDDR** J21 pins 1 - 6, 9 - 10 data direction register
- **JOYSWAP** Exchange joystick ports 1 & 2
- **KEYLEDENA** Keyboard LED control enable
- **KEYLEDREG** Keyboard LED register select (R,G,B channels x 4 = 0 to 11)
- **KEYLEDVAL** Keyboard LED register value (write only)
- **KEYLEFT** Directly read C65 Cursor left key
- **KEYQUEUE** 1 = Typing event queue is non-empty. Write 0 to this bit to flush queue.
- **KEYUP** Directly read C65 Cursor up key
- **KSCNRATE** Physical keyboard scan rate ($00=50MHz, $FF=~200KHz)
- **LJOYA** Rotate inputs of joystick A by 180 degrees (for left handed use)
- **LJOYB** Rotate inputs of joystick B by 180 degrees (for left handed use)
- **M65MODEL** MEGA65 model ID. Can be used to determine the model of MEGA65 a programme is running on, e.g., to enable touch controls on MEGAphone.
- **MALT** ALT key state (immediate; read only).
- **MCAPS** CAPS LOCK key state (immediate; read only).
- **MCTRL** CTRL key state (immediate; read only).
- **MDISABLE** Disable modifiers.
- **MLSHFT** Left shift key state (immediate; read only).
- **MMEGA** MEGA/C= key state (immediate; read only).

- **MODKEYALT** ALT key state at top of typing event queue. 1 = held during event.
- **MODKEYCAPS** CAPS LOCK key state at top of typing event queue. 1 = held during event.
- **MODKEYCTRL** CTRL key state at top of typing event queue. 1 = held during event.
- **MODKEYLSHFT** Left shift key state at top of typing event queue. 1 = held during event.
- **MODKEYMEGA** MEGA/C= key state at top of typing event queue. 1 = held during event.
- **MODKEYRSHFT** Right shift key state at top of typing event queue. 1 = held during event.
- **MODKEYSCRL** NOSCRL key state at top of typing event queue. 1 = held during event.
- **MRSHFT** Right shift key state (immediate; read only).
- **MSCRL** NOSCRL key state (immediate; read only).
- **OSKALT** Display alternate on-screen keyboard layout (typically dial pad for MEGA65 telephone)
- **OSKDEBUG** Debug OSK overlay (WRITE ONLY)
- **OSKDIM** Light or heavy dimming of background material behind on-screen keyboard
- **OSKEN** Enable display of on-screen keyboard composited overlay
- **OSKTOP** 1=Display on-screen keyboard at top, 0=Disply on-screen keyboard at bottom of screen.
- **OSKZEN** Display hardware zoom of region under first touch point for on-screen keyboard
- **OSKZON** Display hardware zoom of region under first touch point always
- **PETSCIIKEY** Top of typing event queue as PETSCII. Write to clear event ready for next.
- **PORTF** PMOD port A on FPGA board (data) (Nexys4 boards only)
- **PORTFDDR** PMOD port A on FPGA board (DDR)
- **POTAX** Read Port A paddle X, without having to fiddle with SID/CIA settings.
- **POTAY** Read Port A paddle Y, without having to fiddle with SID/CIA settings.
- **POTBX** Read Port B paddle X, without having to fiddle with SID/CIA settings.
- **POTBY** Read Port B paddle Y, without having to fiddle with SID/CIA settings.

- **REALHW** Set to 1 if the MEGA65 is running on real hardware, set to 0 if emulated (Xemu) or simulated (ghdl)
- **RST41** Internal 1541 drive reset (1=reset, 0=operate)
- **SDBSH** Enable SD card bitbash mode
- **SDCLK** SD card SCLK
- **SDCS** SD card CS_BO
- **SDDATA** SD card MOSI/MISO
- **SYSCTL** System control flags (target specific)
- **UFAST** C65 UART BAUD clock source: 1 = 7.09375MHz, 0 = 80MHz (VIC-IV pixel clock)
- **VIRTKEY1** Set to $7F for no key down, else specify virtual key press.
- **VIRTKEY2** Set to $7F for no key down, else specify 2nd virtual key press.
- **VIRTKEY3** Set to $7F for no key down, else specify 3nd virtual key press.

CHAPTER 7

45E100 Fast Ethernet Controller

- Overview
- Memory Mapped Registers
- Example Programs

OVERVIEW

The 45E100 is a new and simple Fast Ethernet controller that has been designed specially for the MEGA65 and for 8-bit computers generally. In addition to supporting 100Mbit Fast Ethernet, it is radically different from other Ethernet controllers, such as the RR-NET.

The 45E100 includes four receive buffers, allowing upto three frames to be received while another is being processed, or to allow less frequent processing of interrupts. These receive buffers can be memory mapped, and also directly accessed using the MEGA65's DMA controller. Together with automatic CRC32 checking on reception, and automatic CRC32 generation for transmit, these features considerably reduce the burden on the processor, and make it much simpler to write ethernet-enabled programs.

The 45E100 also supports true full-duplex operation at 100Mbit per second, allowing for total bi-directional throughput exceeding 100Mbit per second. The MAC address is software configurable, and promiscuous mode is supported, as are individual control of the reception of broadcast and multi-cast Ethernet frames.

The 45E100 also supports both transmit and receive interrupts, allowing greatly improved real-world performance. When especially low latency is required, it is also possible to immediately abort the transmission of the current Ethernet frame, so that a higher-priority frame can be immediately sent. These features combine to enable sub-millisecond round trip latencies, which can be of particular value for interactive applications, such as multi-player network games.

Differences to the RR-NET and similar solutions

The RR-NET and other Ethernet controllers for the Commodore™ line of 8-bit home computers generally use an Ethernet controller that was designed for 16-bit PCs, but that also supports a so-called "8-bit mode," which suffers from a number of disadvantages. These disadvantages include the lack of working interrupts, as well as processor intensive access to the Ethernet frame buffers. The lack of interrupts forces programs to use polling to check for the arrival of new Ethernet frames. This, together with the complexities of accessing the buffers results in an Ethernet interface that is very slow, and whose real-world throughput is considerably less than its theoretical 10Mbits per second. Even a Commodore 64 with REU cannot achieve speeds above several tens of kilobytes per second.

In contrast, the 45E100 supports both RX (Ethernet frame received) interrupts and TX (ready to transmit) interrupts, freeing the processor from having to poll the device. Because the 45E100 supports RX interrupts, there is no need for large numbers of receive buffers, which is why the 45E100 requires only two RX buffers to achieve very high levels of performance.

Further, the 45E100 supports direct memory mapping of the Ethernet frame buffers, allowing for much more efficient access, including by DMA. Using the MEGA65's in-

tegrated DMA controller it is quite possible to achieve transfer rates of several megabytes per second – some 100x faster than the RR-NET.

Theory of Operation: Receiving Frames

The 45E100 is simple to operate: To begin receiving Ethernet frames, the programmer needs only to clear the RST and TXRST bits (bit 0 of register $D6E0) to ensure that the Ethernet controller is reset, and then set these bits to 1, to release the controller from the reset state. It will then auto-negotiate connection at the highest available speed, typically 100Mbit, full-duplex.

If you wish to simply poll for the arrival of ethernet frames, check the RXQ bit (bit 5 of $D6E1). If it is set, then there is at least one frame that has been received. To access the next frame that has been received, write $01 to $D6E1, and then $03 to $D6E1. This will rotate the ring of receive buffers, to make the next received frame accessible by the processor. The receive buffer that was previously accessible by the processor is marked free, and the 45E100 will use it to receive another ethernet frame when required.

Because the 45E100 has four receive buffers, it is possible that to process multiple frames in succession by following this procedure. If all receive buffers contain received frames, and the processor has not accepted them, then the RXBLKD signal will be asserted, so that the processor knows that it if any more frames are received, they will be lost. Programmers should take care to avoid this situation. As the 45E100 supports receive interrupts, this is generally easy to manage – but don't underestimate how often ethernet frames can arrive on a 100mbit Fast Ethernet connection: If a sender sends a continuous stream of minimum-length ethernet frames, they can arrive every 6 microseconds or so! While, it is unlikely that you will have to deal with such a high rate of packet reception, you should anticipate the need to process packets at least every milli-second. In particular, a once-per-frame CIA or raster IRQ may cause some packets to be lost, more than three arrive in a 16 – 20 ms video frame. The RXBLKD signal can be used to determine if this situation is likely to have occurred. But note that it indicates only when all receive buffers are occupied, not if any further frames arrived while there were no free receive buffers.

The receive buffers are 2KB bytes each, and can each hold only one received ethernet frame at a time. This is different to some ethernet controllers that use their total receive buffer memory as a simple ring buffer. The reason for this is to keep the mechanism for programmers as simple as possible. By having the fixed buffers, it means that the controller can memory map the received ethernet frames in exactly the same location each time, making it possible to write much simpler receiver programs, because the location of the received ethernet frames can be assumed to be constant.

The structure of a receive buffer containing an ethernet frame is quite simple: The first two bytes indicate the length of the received frame. The frame then follows immediately. The effective Maximum Transport Unit (MTU) length is 2,042 bytes, as the last four bytes are occupied by the CRC32 checksum of the received ethernet frame. The layout of the receive buffers is thus as follows:

HEX	DEC	Length	Description
0000	0	1	The low byte of the length of the received ethernet frame.
0001	1	1	The lower four bits contain the upper bits of the length of the received ethernet frame. Bit 4 is set if the received ethernet frame is a multi-cast frame. Bit 5 if it is a broadcast frame. Bit 6 is set if the frame's destination address matches the 45E100's programmed MAC address. Bit 7 is set if the CRC32 check for the received frame failed, i.e., that the frame is either truncated or was corrupted in transit.
0002 - 07FB	2 - 2,043	2,042	The received frame. Frames shorter than 2,042 bytes will begin at offset 2.
07FC - 07FF	2,044 - 2,047	4	Reserved space for holding the CRC32 code during reception. The CRC32 code is, however, always located directly after the received frame, and thus will only occupy this space if the received frame is more than 2,038 bytes long. "

Because of the very rapid rate at which Fast Ethernet frames can be received, a programmer should use the receive interrupt feature, enabled by setting RXQEN (bit 7 of $D6E1). Polling is possible as an alternative, but is not recommended with the 45E100, because at the 100Mbit Fast Ethernet speed, packets can arrive as often as every 5 microseconds. Fortunately, at the MEGA65's 40MHz full speed mode, and using the 20MB per second DMA copy functionality, it is possible to keep up with such high data rates.

Accessing the Ethernet Frame Buffers

Unlike on the RR-NET, the 45E100's ethernet frame buffers are able to be memory mapped, allowing rapid access via DMA or through assembly language programs. It is also possible to access the buffers from BASIC with some care.

The frame buffers can either be accessed from their natural location in the MEGA65's extended address space at address $FFDE800 - $FFDEFFF, or they can be mapped into the normal C64/C65 $D000 I/O address space. Care must be taken as mapping the ethernet frame buffers into the $D000 I/O address space causes all other I/O devices to unavailable during this time. Therefore CIA-based interrupts MUST be disabled before doing so, whether using BASIC or machine code. Therefore when programming in assembly language or machine code, it is recommended to use the natural location, and to access this memory area using one of the three mechanisms for accessing extended address space, which are described in *the MEGA65 Book*, Accessing memory beyond the 1MB point (subsection J).

The method of disabling interrupts differs depending on the context in which a program is being written. For programs being written using C64-mode's BASIC 2, the following will work:

```
POKE56333,127: REM DISABLE CIA TIMER IRQS
```

While for MEGA65's BASIC 65, the following must instead be used, because a VIC-III raster interrupt is used instead of a CIA-based timer interrupt:

```
POKE53274,0: REM DISABLE VIC-II/III/IV RASTER IRQS
```

Once this has been done, the I/O context for the ethernet controller can be activated by writing $45 (69 in decimal, equal to the character 'E' in PETSCII)) and $54 (84 in decimal, equal to the character 'T' in PETSCII) into the VIC-IV's KEY register ($D02F, 53295 in decimal), for example:

```
POKE53295,ASC("E"):POKE53295,ASC("T")
```

At this point, the ethernet RX buffer can be read beginning at location $D000 (53248 in decimal), and the TX buffer can be written to at the same address. Refer to 'Theory of Operation: Receiving Frames' above for further explanation on this.

Once you have finished accessing the ethernet frame buffer, you can restore the normal C64, C65 or MEGA65 I/O context by writing to the VIC-III/IV's KEY register. In most cases, it will make the most sense to revert to the MEGA65's I/O context by writing $47 (71 decimal) in and $53 (83 in decimal) to the KEY register, for example:

```
POKE53295,ASC("G"):POKE53295,ASC("S")
```

Finally, you should then re-enable interrupts, which will again depend on whether you are programming from C64 or C65-mode. For C64-mode:

```
POKE56333,129
```

For C65-mode it would be:

```
POKE53274,129
```

Theory of Operation: Sending Frames

Sending frames is similarly simple: The program must simply load the frame to be transmitted into the transmit buffer, write its length into TXSZLSB and TXSZMSB registers, and then write $01 into the COMMAND register. The frame will then begin to transmit, as soon as the transmitter is idle. There is no need to calculate and attach an ethernet CRC32 field, as the 45E100 does this automatically.

Unlike for the receiver, there is only one frame buffer for the transmitter (this may be changed in a future revision). This means that you cannot prepare the next frame until the previous frame has already been sent. This slightly reduces the maximum data throughput, in return for a very simple architecture.

Also, note that the transmit buffer is write-only from the processor bus interface. This means that you cannot directly read the contents of the transmit buffer, but must load values "blind". Finally, the 45E100 allows you to send ethernet.

Advanced Features

In addition to operating as a simple and efficient ethernet frame transceiver, the 45E100 includes a number of advanced features, described here.

Broadcast and Multicast Traffic and Promiscuous Mode

The 45E100 supports filtering based on the destination Ethernet address, i.e., MAC address. By default, only frames where the destination Ethernet address matches the ethernet address programmed into the MACADDR1 - MACADDR6 registers will be received. However, if the MCST bit is set, then multicast ethernet frames will also be received. Similarly, setting the BCST bit will allow all broadcast frames, i.e., with MAC address ff:ff:ff:ff:ff:ff, to be received. Finally, if the NOPROM bit is cleared, the 45E100 disables the filter entirely, and will receive all valid ethernet frames.

Debugging and Diagnosis Features

The 45E100 also supports several features to assist in the diagnosis of ethernet problems. First, if the NOCRC bit is set, then even ethernet frames that have invalid CRC32 values will be received. This can help debug faulty ethernet devices on a network.

If the STRM bit is set, the ethernet transmitter transmits a continuous stream of debugging frames supplied via a special high-bandwidth logging interface. By default, the 45E100 emits a stream of approximately 2,200 byte ethernet frames that contain compressed video provided by a VIC-IV or compatible video controller that supports the MEGA65 video-over-ethernet interface. By writing a custom decoder for this stream of ethernet frames, it is possible to create a remote display of the MEGA65 via ethernet. Such a remote display can be used, for example, to facilitate digital capture of the display of a MEGA65.

The size and content of the debugging frames can be controlled by writing special values to the COMMAND register. Writing $F1 allows the selection of frames that are 1,200 bytes long. While this reduces the performance of the debugging and streaming features, it allows the reception of these frames on systems whose ethernet controllers cannot be configured to receive frames of 2,200 bytes.

If the STRM bit is set and bit 2 of $D6E1 is also set, a compressed log of instructions executed by the 45gs02 CPU will instead be streamed, if a compatible processor is connected to this interface. This mechanism includes back-pressure, and will cause the 45gs02 processor to slowdown, so that the instruction data can be emitted. This

typically limits the speed of the connected 45gs02 processor to around 5MHz, depending on the particular instruction mix.

Note also that the status of bit 2 of $D6E1 cannot currently be read directly. This may be corrected in a future revision.

Finally, if the video streaming functionality is enabled, this also enables reception of synthetic keyboard events via ethernet. These are delivered to the MEGA65's Keyboard Complex Interface Adapter (KCIA), allowing full remote interaction with a MEGA65 via its ethernet interface. This feature is primarily intended for development.

MEMORY MAPPED REGISTERS

The 45E100 Fast Ethernet controller is a MEGA65-specific feature. It is therefore only available in the MEGA65 I/O context. This is enabled by writing $53 and then $47 to VIC-IV register $D02F. If programming in BASIC, this can be done with:

```
POKE53295,ASC("G"):POKE53295,ASC("S")
```

The 45E100 Fast Ethernet controller has the following registers

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D6E0	55008	TXIDLE	RXBLKD	-	RCENABLED	DRXDV	DRXD	TXRST	RST
D6E1	55009	RXQEN	TXQEN	RXQ	TXQ	STRM	RXBF		-
D6E2	55010	TXSZLSB							
D6E3	55011	TXSZMSB							
D6E4	55012	COMMAND							
D6E5	55013	RXPH		MCST	BCST	TXPH		NOCRC	NOPROM
D6E6	55014	MIIMPHY			MIIMREG				
D6E7	55015	MIIMVLSB							
D6E8	55016	MIIMVMSB							
D6E9	55017	MACADDR1							
D6EA	55018	MACADDR2							
D6EB	55019	MACADDR3							
D6EC	55020	MACADDR4							
D6ED	55021	MACADDR5							
D6EE	55022	MACADDR6							

- **BCST** Accept broadcast frames
- **COMMAND** Ethernet command register (write only)
- **DRXD** Read ethernet RX bits currently on the wire
- **DRXDV** Read ethernet RX data valid (debug)

- **MACADDRX** Ethernet MAC address
- **MCST** Accept multicast frames
- **MIIMPHY** Ethernet MIIM PHY number (use 0 for Nexys4, 1 for MEGA65 r1 PCBs)
- **MIIMREG** Ethernet MIIM register number
- **MIIMVLSB** Ethernet MIIM register value (LSB)
- **MIIMVMSB** Ethernet MIIM register value (MSB)
- **NOCRC** Disable CRC check for received packets
- **NOPROM** Ethernet disable promiscuous mode
- **RCENABLED** (read only) Ethernet remote control enable status
- **RST** Write 0 to hold ethernet controller under reset
- **RXBF** Number of free receive buffers
- **RXBLKD** Indicate if ethernet RX is blocked until RX buffers freed
- **RXPH** Ethernet RX clock phase adjust
- **RXQ** Ethernet RX IRQ status
- **RXQEN** Enable ethernet RX IRQ
- **STRM** Enable streaming of CPU instruction stream or VIC-IV display on ethernet
- **TXIDLE** Ethernet transmit side is idle, i.e., a packet can be sent.
- **TXPH** Ethernet TX clock phase adjust
- **TXQ** Ethernet TX IRQ status
- **TXQEN** Enable ethernet TX IRQ
- **TXRST** Write 0 to hold ethernet controller transmit sub-system under reset
- **TXSZLSB** TX Packet size (low byte)
- **TXSZMSB** TX Packet size (high byte)

COMMAND register values

The following values can be written to the COMMAND register to perform the described functions. In normal operation only the STARTTX command is required, for example, by performing the following **POKE**:

```
POKE55012,1
```

HEX	DEC	Signal	Description
00	0	STOPTX	Immediately stop transmitting the current ethernet frame. Will cause a partially sent frame to be received, most likely resulting in the loss of that frame.
01	1	STARTTX	Transmit packet
D0	208	RXNORMAL	Disable the effects of RXONLYONE
D4	212	DEBUGVIC	Select VIC-IV debug stream via ethernet when $D6E1.3 is set
DC	220	DEBUGCPU	Select CPU debug stream via ethernet when $D6E1.3 is set
DE	222	RXONLYONE	Receive exactly one ethernet frame only, and keep all signals states (for debugging ethernet sub-system)
F1	241	FRAME1K	Select 1KiB frames for video/cpu debug stream frames (for receivers that do not support MTUs of greater than 2KiB)
F2	242	FRAME2K	Select 2KiB frames for video/cpu debug stream frames, for optimal performance.

EXAMPLE PROGRAMS

Example programs for the ethernet controller exist in imperfect states for in the MEGA65 Core repository on github in the src/tests and src/examples directories.

If you wish to use the ethernet controller for TCP/IP traffic, you may wish to examine the port of WeeIP to the MEGA65 at `https://github.com/mega65/mega65-weeip`. The code that controls the ethernet controller is located in `eth.c`.

CHAPTER 8

451027 Multi-Function I/O Controller

- Overview
- F011-compatible Floppy Controller
- SD card Controller and F011 Virtualisation Functions
- Touch Panel Interface
- Audio Support Functions
- Miscellaneous I/O Functions

OVERVIEW

The 45IO27 is a multi-purpose I/O controller that incorporates the functions of the C65's F011 floppy controller, together with the MEGA65's SD card controller interface, and a number of other miscellaneous I/O functions.

Each of these major functions is covered in a separate section of this chapter.

F011-COMPATIBLE FLOPPY CONTROLLER

The MEGA65 computer is one of the very few modern computers that still includes first-class support for magnetic floppy drives. It includes a floppy controller that is backwards compatible with the C65's F011D floppy drive controller.

However, unlike the F011D, the MEGA65's floppy disk controller supports HD and ED media, and similar to the 1541 floppy drive, it also supports variable data rates, so that a determined user could develop disk formats that store more data, include robust copy protection schemes, or both.

Other disk formats, such as the GCR scheme used by the 1541 and 1571, or the track-at-once MFM scheme used by the Amiga™ family of computers can also be used, with varying amounts of effort. This is possible via the track DMA functionality that is jointly provided by the 45IO27 and the 45GS02 processor. These two devices work together to allow reading or writing raw flux signals from and to diskettes, thus allowing the reading or reproduction of practically any disk format.

Also, for Amiga™ diskettes, there is experimental special circuitry in the 45IO27 to ease the reading of these diskettes. Specifically, the 45IO27 is able to detect the unique sector sync signals used on Amiga™ diskettes, and read the data sectors. However, it does not retrieve the 16 bytes per sector of operating-system specific data, nor does it perform the de-interleaving of the separated vectors of odd and even bits from the order that the Amiga™ computer writes them to disk. Use of this special circuitry is optional. That is, the track DMA functionality can also be used to read Amiga™ diskettes.

Multiple Drive Support

Like the C65's F011 floppy drive controller, the 45IO27 supports up to 8 drives. The first two of those drives, drive 0 and drive 1, are assumed to be connected to a standard 34-pin floppy cable, the same as used in standard PCs, i.e., with a twist in the cable to allow the use of two unjumpered drives.

As is described in later sections, it is possible to switch drive 0 and drive 1's position, without having to change cabling. Similarly, either or both of the first two drives may

reference a real floppy drive, a D81 disk image stored on an attached SD card, or redirected to the floppy drive virtualisation service, so that the sector accesses can be handled by a connected computer, e.g., as part of a comfortable and efficient cross-development environment.

The remaining six drives are supported only in conjunction with a future C1565-compatible external drive port.

Buffered Sector Operations

The 45IO27 support two main modes of reading sectors from a disk: byte-by-byte, and via a memory-mapped sector buffer.

The byte-by-byte mechanism consists of having a loop wait for the DRQ signal to be asserted, and then reading the byte of data from the DATA register ($D087).

The memory-mapped sector buffer method consists of waiting for the BUSY flag to clear, indicating that the entire sector has been read, and then directly accessing the sector buffer located at $FFD6C00 - $FFD6DFF. Care should be taken to ensure that the BUFSEL signal (bit 7 of $D689) is cleared, so that the floppy sector buffer is visible, rather than the SD card sector buffer for programs other than the Hypervisor. This is because only the Hypervisor has access to the full 4KB SD controller buffer space: Normal programs see either the floppy sector buffer or the SD card sector buffer repeated 8 times between $FFD6000 and $FFD6FFF.

Alternatively, the sector buffer can be mapped at $DE00 - $DFFF, i.e., in the 4KB I/O area, by writing the $81 to the SD command register at $D680. This will hide any I/O peripherals that are otherwise using this area, e.g., from cartridges, or REU emulation. This function can be disabled again by writing $82 to the SD command register. As with the normal sector buffer memory mapping at $FFD6xxx, the BUFSEL signal (bit 7 of $D689) affects whether the FDC or the SD card sector buffer is visible, for software not running in Hypervisor mode. Note that if you use the Matrix Mode / serial monitor interface to inspect the contents of the sector buffer, that this occurs in the Hypervisor context, and so the BUFSEL signal will be ignored, and the full 4KB buffer will be visible.

The memory-mapped sector buffer has the advantage that it can be accessed via DMA, allowing for very efficient copies. Also, it allows for loading a sector to occur in the background, while your program gets on with more interesting things in the meantime.

Reading Sectors from a Disk

There are several steps that you must follow in order to successfully read a sector from a disk. If you follow these instructions, your code will work with both physical disks, as well as D81 disk images that exist on the SD card:

- First, enable the motor and select the appropriate drive. The F011 supports upto 8 physical drives, although it is rare for more than two to be physically connected. To enable the motor, write $60 to $D080. You should then write a

SPINUP command ($20) to $D081, and wait for the BUSY flag (bit 7 of $D082) to clear. The drive is now spinning at speed, and ready to service requests.

- Next, select the correct side of the disk by either setting or clearing the SIDE1 flag (bit 3 of $D080). This takes effect immediately.
- Third, use the step-in and step-out commands (writing $10 and $18 to $D081) as required to move the head to the correct track. Again, after each command, you should wait for the BUSY flag (bit 7 of $D082) to clear, before issuing the next command.

  Note that you can check if the head is at track 0 by checking the TRACK0 flag, but there is no fool-proof way to know if you are on any other specific track. You can use the registers at $D6A3 - $D6A5 to see the track, sector and side value from the last sector header which passed under the head to make an informed guess as to which track is currently selected. Note that this only works for real disks, as disk images do not spin under the read head. Also note that it is possible for tracks to contain sectors which purposely or accidently have incorrect track numbers in the sector headers.
- Fourth, you need to load the desired track, sector and side number into the TRACK, SECTOR and SIDE registers ($D084, $D085 and $D086, respectively). The FDC is now primed ready to read a sector.
- Fifth, you should write an appropriate read command value into $D081. This will normally be $40 (64). You then wait for the RDREQ signal ($D083, bit 7) to go high, to indicate that the sector has been found. You then either wait for each occassion when DRQ goes high, and read byte-by-byte in such a loop, or wait for the BUSY flag to clear and the DRQ and EQ flags to go high, which indicates that the complete sector has been read into the buffer.

Track Auto-Tune Function

The 45IO27 also includes a track "auto-tune" function, which is enabled by clearing bit 7 of $D696. That function reads the sector headers to determine which track the head is currently over, and steps the head in or out to try to get to the correct track. Auto-tune is enabled by default.

Sector Skew and Target Any Mode

It is also worth noting that the TARGANY signal can be asserted to tell the floppy controller to simply read the next sector that passes under the head. This applies only when using real floppy disks, where it offers the considerable advantage of letting you read the sectors in the order in which they exist on the disk. This allows you to read a track at once, without having to wait for the index hole to pass by, or having to know which sector will next pass under the head.

For example, the C65 DOS formats disks using a skew factor of 7, while PCs may use a different skew-factor. If you don't know the skew factor of the disk, you may

schedule the reading of the sectors on the track in a sub-optimal order. This can result in transfer rates as low as 5 sectors per second, compared with the optimal case of 50 sectors per second. Thus with either correct sector order, or using the target any mode, it is possible to read approximately two full tracks per second, i.e., two sides $\times$ two tracks, or approximately 20KB/second on DD disks, or double that on HD disks, at around 40KB/second. This compares very favourably with the C65 DOS loading speed, which is typically nearer 1KB/sec in C64-mode.

Disk Layout and 1581 Logical Sectors

The 1581 disk format is unusual in that the physical sectors on the disk are a different size of the size of the data blocks that it presents to the user. Specifically, the disks use 512 byte sectors, while the 1581 (and C65) DOS present 256 byte data blocks. Two blocks are stored in each physical sector. Also, the physical track numbers are from 0 to 79, while the logical track numbers of the DOS are 1 to 80. Physical sectors are also numbered from 1 to 10, while logical block numbers begin are 0 to 39.

This means that if you want to find a 1581 logical sector, you need to know which physical sector it will be found in. To determine the physical sector that contains a block, you first subtract one from the track number, and then divide the sector number by two. Logical sectors 0 to 19 of each track are located in physical sectors 1 to 10 on the first side of the disk. Logical sectors 20 to 39 are located in physical sectors 1 to 10 on the reverse side of the disk.

Thus we can map a some logical track and sector t,s to the physical track, side and sector as follows:

$track = t - 1$

$sector = (s/2) + 1, IFFs < 20, ELSE = ((s - 20)/2) + 1$

$side = 0IFFsector < 20$

It is also worth noting that the 45IO27 is capable of reading from tracks beyond track 80, provided that the disk drive is capable of this. Almost all 3.5 inch floppy drives are capable of reading at least one extra track, as historically manufacturers of floppy disks stored information about the disk on the 81st track. In our experience almost all drives will also be able to access an 82nd track.

FD2000 Disks

The CMD™ FD2000™ high-density 3.5" disk drives for Commodore™ computers use an unusual disk layout that is quite different from PCs: They use 10 sectors, the same as on 720KB double-density (DD) disks, but double the *sector size* from 512 bytes to 1,024 bytes. The 45IO27 does not currently support these larger sectors for buffered sector operation. At least read-only support is planned to be added via a core update in the future. However, FD2000™ and FD4000™ diskettes can be read and written using the track DMA functionality.

Also, for creating new diskettes, the 45IO27 *does* already support high-density disks and drives, with much higher capacities than the FD2000 was able to support.

High-Density and Variable-Density Disks

The 45IO27 supports variable data rates, allowing the use of HD drives and media, with a flexible approach to disk formats to support user experimentation, and the easy manipulation of high-capacity software distribution formats.

You are really only limited by your imagination, available time, and the limited number of people who are still interested in inserting a floppy disk into their computer!

The standard high-density (HD) disk format is "1.44MB", using 18 sectors per track over 80 tracks. This results in 80 tracks × 18 sectors × 2 sides = 2,880 sectors. As each sector is 512 bytes, this corresponds to 1,440KB. This leads us into the interesting wonderland of "floppy disk marketing megabytes," a phenomena which long predates SD card and hard drive manufacturers using 1,000,000 byte megabytes.

Curiously for floppy disks, the 1,024,000 byte "megabyte" was used, i.e., "1MB" = 1KB × 1KB, that is a strange hybrid of binary and decimal conventions. Perhaps it was because the previous standard was 720KB, and they thought people would think it odd if double 720KB was 1.41MB, and complain about the missing kilo-bytes. We will continue to use the 1,024KB = 1,000KB floppy disk marketing mega-byte for consistency with this historical inconsistency.

However, HD floppy disks are fundamentally capable of holding much more than 1.44MB. For example, the FD2000 stored 1.6MB by using double-sized sectors to squeeze the equivalent of 20 sectors per track, and the Amiga went further by using track-at-once writing to fit 22 sectors per track. Both these formats used a constant data rate over all tracks, and thus a constant number of sectors per track.

However, the circumference of the tracks on a 3.5" floppy disk vary quite a lot: The inner track has a diameter of around 2.5cm, while the outside track is 1.6× longer. The 1.44MB disk format is designed so that the data is reliably stored on those shorter inner tracks. This means that we should be able to fit 160% more data on the outer-most track compared with the inner-most track, subject to a number of terms and conditions imposed by The Laws of Physics, the design of floppy drive electronics, the quality of media being used and various other annoying things. Because of this variability and uncertainty, the MEGA65's floppy controller supports fully variable data rate on a track-by-track basis.

Track Information Blocks

To support variable data rates, the 45GS27 supports the use of *Track Information Blocks* (TIBs) that contain information on the data rate and encoding used on the track. This allows users to experiment with various densities on various tracks, and yet have the disks function automatically for buffered sector operations.

The Track Information Block is automatically created when using the automatic track format function, but must be manually created if using unbuffered formatting. The TIB itself consists of the following data:

1. 3× $A1 Sync bytes (written with clock byte $FB)
2. $65 MEGA65 Track Information Block marker (written with clock byte $FF, as are all following bytes in the block)
3. The track number
4. The data rate divisor, in the same format as $D6A2, i.e., data rate = 40.5MHz / value.
5. Track encoding information: Bit 7 = Track-at-once flag, 1 = no inter-sector gaps (Amiga style), 0 = with inter-sector gaps (normal), Bit 6 = data encoding, 0 = MFM, 1=RLL2,7. Other bits are reserved, and should be 0 when written.
6. Sector count, i.e., number of sectors on the track.
7. CRC byte 1, using the normal floppy disk CRC algorithm.
8. CRC byte 2, using the normal floppy disk CRC algorithm.

The Track Information Block is always written a the data rate for a 720KB Double-Density disk, so that they can be present on any disk. Writing the Track Information Block and start-of-track gaps at the DD data rate also ensures that at very high data rates, the head still has sufficient time to switch to write mode, thus avoiding one of the many problems that arise when writing data at very high data rates.

If formatting disks unbuffered, it is the programmer's responsibility to switch the data rate after having written the Track Information Block, and several more bytes to allow the floppy encoding pipeline to flush out the last byte of the Track Information Block. This is all automatically managed if using the automatic track formatting function.

The inclusion of the TIB allows users to play and explore the possibilities of different data rates on different drives and media, while still being automatically readable in all MEGA65s, because the TIB allows the controller to switch to the correct data rate and encoding. It is likely that over time somewhat standardised formats will develop, quite likely in the range of 2MB to 3.5MB - thus approaching the capacity of ED media in ED drives, without the need for those drives or media.

Formatting Disks

Formatting disks is now possible with the 45IO27, either unbuffered or fully-automatic. To format a track issue one of the following commands to $D081:

- $A0 - Automatic format, with inter-sector gaps, and write pre-compensation disabled.
- $A1 - Manual format, write-precompensation disabled.

- $A4 - Automatic format, with inter-sector gaps, and write pre-compensation enable.
- $A5 - Manual format, write-precompensation enabled.
- $A8 - Automatic format, Amiga-style track-at-once, and write pre-compensation disabled.
- $AC - Automatic format, Amiga-style track-at-once, and write pre-compensation enable.

Manual formatting is not recommended, unless mastering track-at-once formatted disks for software distribution, because of the relative complexity of doing so. Also, at the higher data rates, bytes have to be delivered to the floppy controller as often as every 20 cycles, which requires considerable care when writing the format routine. For more information on manual formatting tracks, refer to the C64 Specifications Manual or the C65 ROM DOS source code, for examples of manual formatting.

The automatic modes, in contrast, format a track with a single command, and are thus much easier to use, and are recommended for general use. Write pre-compensation should normally be enabled, as it is required at higher data rates, and does not cause problems at lower data rates.

Write Pre-Compensation

Write pre-compensation is a family of algorithms used when writing high data-rate signals to floppy disks. It is used to anticipate and cancel out the predictable component of timing variation of magnetic recording. There are a variety of sources of this timing variation, which have been the subject of PhD theses, and a lot of proprietary research by hard drive manufacturers. What is important for us to understand is that adjacent pulses (really magnetic inversions) get pushed together, if they are surrounded by longer pulses, or tend to spread apart if surrounded by shorter pulses.

There are also other fascinatingly complex and difficult to predict factors, that cause things such as the "negative shift of mid-length pulses", "inverse F-distribution of pulse arrival times" and goodness knows what else. But we shall leave those to the hard drive manufacturers. We limit ourselves to the data pattern induced effect described in the previous paragraph.

The 45GS27 supports two tunable coefficients for small and large corrections to this, which are used with an internal look-up table. However, this is all automatically handled if you enable write pre-compensation. This allows data rates that much more closely approach the expected limit of HD media, although due to the other horrors of magnetic media recording alluded to above, the actual limit is not reached.

Buffered Sector Writing

The 45IO27 can write to disk images that are located on the SD card, or when using virtualised disk access.

To write a sector, you follow a similar process to reading, except that you write $84 to the command byte instead of $40. The $80 indicates a write, and the $04 activates write-precompensation. This is important when writing to real floppy disks, especially HD and ED disks. Write-precompensation causes bits to be written slightly early or slightly late, using an algorithm that models how the magnetic domains on a disk tend to move after being written.

If you do not wish to use the sector buffer, but instead provide each byte one at a time during the write operation, you must add $01 to the command code. However, this is not recommended on the MEGA65, because when writing to the SD card or using virtualised disk images the entire sector operation can happen instantaneously from the perspective of your program. This means that it is not possible to supply data reliably when in this mode. Thus apart from being less convenient, it is also less reliable.

Once a write operation has been triggered, the DRQ signal indicates when you should provide the next byte if performing a byte-by-byte write. Otherwise, it is assumed that you will have pre-filled the sector buffer with the complete 512 bytes of data required.

To write to disks that contain Track Information Blocks, you should first wait for the TIB to be read when changing tracks. This is done by waiting for $D6A9 (sectors per track from the TIB) to contain a non-zero value.

Floppy Track DMA

As previously described, the 45IO27 connects to the 45GS02 CPU, and specifically its internal DMA controller, to provide a simple mechanism for reading and writing the raw magnetic flux transitions of floppy disks. In addition to allowing writing softwar that can read any possible disk format, it also allows for the writing and duplication of almost any disk format. I say almost, because it is possible for diskettes to be written using special machines that have capabilities that exceed that of the floppy drive mechanism of the MEGA65. Fat tracks is one example of this. Another example is where the magnetic flux transitions are placed and/or spaced in a manner that prevents a normal floppy drive from reproducing them exactly.

Using Floppy Track DMA

The following DMA options tell the DMAgic controller to read or write raw floppy flux values instead of memory location. In this mode, the DMAgic controller waits for each successive pulse interval to be received from the 45IO27. Thus the data that it reads (or writes) is the duration between each successive flux inversion, in units of 50ns.

- $0D - Write raw flux intervals. Set DMA to COPY mode to use this.
- $0E - Read raw flux intervals, ignoring implausibly long intervals. Set DMA mode to FILL to use this.
- $0F - Read raw flux intervals, returning implausibly long intervals as $FF. Set DMA mode to FILL to use this.

See floppytest.c in the mega65-tools repository of the MEGA65 github site at https://github.com/mega65-tools for example code that reads tracks using this functionality.

Understanding the Limitations of Floppy Drives

When writing raw flux transitions, care must be taken to understand the limitations of standard floppy drives. There are three key factors to be considered:

- No two transitions can be placed too close togther. The drive will simply refuse to write them if they are below some distance apart. Also, even if you do manage to write them, the read filtering circuitry of the floppy drive will merge pulses that are too close together.
- No two transitions can be placed too far apart. In this case, the drive will happily write them, but the read filter circuitry will start getting worried if it doesn't see a pulse for a while, and will start thinking that the background noise is real signals, and introduce false pulses into the read stream that are not really there on the disk.
- Magnetic pulses move on the disk when you write them! The physics behind this is really complex, and depends on the sequence of distances between successive pulses, the data that was previously written to the track, and, it feels like, also the phase of the moon and colour of your underpants. There are some general rules of thumb that can be used at "typical" pulse distance intervals to partially mitigate this. These rules are called Write Pre-compensation, where the pulses are moved before writing, so that after writing they end up in the right place. The 45IO27 already implements write pre-compensation for HD formatted disks. The effects of moving pulses is especially pronounced for pulses that are closer together.
- The higher numbered tracks are shorter. This means that the pulses have to be further apart in time, to be the same distance apart on a track. This is why the 1541 fits less data on higher numbered tracks. This is especially pronounced at higher data rates, where the "magnetic resolution" of the disks becomes an issue. To make matters worse, the strength of the magnetic signals is proportional to the speed of the track going past. This means for the higher numbered tracks that are nearer the middle of the disk, not only is the "magnetic resolution" lower due to the shorter tracks, the linear velocity of the disk under the head is also lower, resulting in weaker signals.

It is the combination of these factors that will tend to limit how densly you can pack data onto a floppy disk - together of course with the quality and condition of the diskette and disk drive. The MEGA65's HD+ disk formats that are able to fit around 3MiB on a standard 1.44MiB diskette take special care to manage these factors. This is why those formats use many different density zones, as well as using RLL27 encoding instead of MFM encoding, because it increases the minimum distance between pulses.

F011 Floppy Controller Registers

The following are the set of F011 compatibility registers of the 45IO47. Note that registers related to the use of SD card based storage are found in the corresponding section below.

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D080	53376	IRQ	LED	MOTOR	SWAP	SIDE	DS		
D081	53377	WRCMD	RDCMD	FREE	STEP	DIR	ALGO	ALT	NOBUF
D082	53378	BUSY	DRQ	EQ	RNF	CRC	LOST	PROT	TK0
D083	53379	RDREQ	WTREQ	RUN	WGATE	DISKIN	INDEX	IRQ	DSKCHG
D084	53380	TRACK							
D085	53381	SECTOR							
D086	53382	SIDE							
D087	53383	DATA							
D088	53384	CLOCK							
D089	53385	STEP							
D08A	53386	PCODE							

- **ALGO** Selects reading and writing algorithm (currently ignored).
- **ALT** Selects alternate DPLL read recovery method (not implemented)
- **BUSY** F011 FDC busy flag (command is being executed) (read only)
- **CLOCK** Set or read the clock pattern to be used when writing address and data marks. Should normally be left $FF
- **COMMAND** F011 FDC command register
- **CRC** F011 FDC CRC check failure flag (read only)
- **DATA** F011 FDC data register (read/write) for accessing the floppy controller's 512 byte sector buffer
- **DIR** Sets the stepping direction (inward vs
- **DISKIN** F011 Disk sense (read only)
- **DRQ** F011 FDC DRQ flag (one or more bytes of data are ready) (read only)
- **DS** Drive select (0 to 7). Internal drive is 0. Second floppy drive on internal cable is 1. Other values reserved for C1565 external drive interface.
- **DSKCHG** F011 disk change sense (read only)
- **EQ** F011 FDC CPU and disk pointers to sector buffer are equal, indicating that the sector buffer is either full or empty. (read only)
- **FREE** Command is a free-format (low level) operation
- **INDEX** F011 Index hole sense (read only)

- **IRQ** The floppy controller has generated an interrupt (read only). Note that interrupts are not currently implemented on the 45GS27.
- **LED** Drive LED blinks when set
- **LOST** F011 LOST flag (data was lost during transfer, i.e., CPU did not read data fast enough) (read only)
- **MOTOR** Activates drive motor and LED (unless LED signal is also set, causing the drive LED to blink)
- **NOBUF** Reset the sector buffer read/write pointers
- **PCODE** (Read only) returns the protection code of the most recently read sector. Was intended for rudimentary copy protection. Not implemented.
- **PROT** F011 Disk write protect flag (read only)
- **RDCMD** Command is a read operation if set
- **RDREQ** F011 Read Request flag, i.e., the requested sector was found during a read operation (read only)
- **RNF** F011 FDC Request Not Found (RNF), i.e., a sector read or write operation did not find the requested sector (read only)
- **RUN** F011 Successive match. A synonym of RDREQ on the 45IO47 (read only)
- **SECTOR** F011 FDC sector selection register
- **SIDE** Directly controls the SIDE signal to the floppy drive, i.e., selecting which side of the media is active.
- **STEP** Writing 1 causes the head to step in the indicated direction
- **SWAP** Swap upper and lower halves of data buffer (i.e. invert bit 8 of the sector buffer)
- **TK0** F011 Head is over track 0 flag (read only)
- **TRACK** F011 FDC track selection register
- **WGATE** F011 write gate flag. Indicates that the drive is currently writing to media. Bad things may happen if a write transaction is aborted (read only)
- **WRCMD** Command is a write operation if set
- **WTREQ** F011 Write Request flag, i.e., the requested sector was found during a write operation (read only)

The following registers apply to the 45IO27 only, i.e., are MEGA65 specific:

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D6A0	54944	DENSITY	DBGMO-TORA	DBGMO-TORA	DBGDIR	DBGDIR	DBGW-DATA	DBGW-GATE	DBGW-GATE

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D6A2	54946	DATARATE							

- **DATARATE** Set number of bus cycles per floppy magnetic interval (decrease to increase data rate)
- **DBGDIR** Control floppy drive STEPDIR line
- **DBGMOTORA** Control floppy drive MOTOR line
- **DBGWDATA** Control floppy drive WDATA line
- **DBGWGATE** Control floppy drive WGATE line
- **DENSITY** Control floppy drive density select line

SD CARD CONTROLLER AND F011 VIRTUALISATION FUNCTIONS

For those situations where you do not wish to use real floppy disks, the 45IO27 supports two complementary alternative modes:

- SD card Based Disk Image Access.
- Virtualised Disk Image Access.

This is in addition to providing direct access to a dual-bus SD card interface.

SD card Based Disk Image Access

The 45IO27 is both a floppy drive and SD card controller. This enables it to transparently allow access to D81 disk images stored on the SD card. Further, because the controller is combined, it is possible to still have the floppy drive step and spin as though it were being used, providing considerable atmosphere and sense of realism, even when using disk images.

The 45IO27 supports both 800KB standard D81 disk images, as well as 64MB "MEGA Images". While an operating system may impose restrictions based on the name of a file, the 45IO27 is blind to these requirements. Instead, it requires only that a contiguous 800KB or 64MB of the SD card is used to contain a disk image.

When a disk image is enabled, the corresponding set of sectors on the SD card are effectively placed under user control, and the operating system is no longer able to prevent the reading or writing of any of those sectors. Thus you should never enable access to an image that is shorter than the required size, as it will otherwise allow the user to unwittingly or maliciously access and/or modify data that is not part of the image file.

For the same reason, only the hypervisor can change the sector number where a disk image starts (the D?STARTSEC? signals), or allow the use of disk images instead of the real floppy drive (USEREAL0 and USEREAL1 signals). Once the Hypervisor has set the start sector of a disk image, and cleared the USEREAL0 or USEREAL1 signal, the user can still controll whether an access will go to the real floppy drive or to the disk image by respectively clearing or setting the appropriate signal. For drive 0, this is D0IMG, and for drive 1, it is D1IMG.

There are also signals to control whether a disk image is an 800KB D81 image or a 64MB MEGA Disk image, and whether a disk image is present, and whether it is write protected. These are all located in the $D68B register. Because of the ability of manipulation of these registers to corrupt or improperly access data, these signals are all read-only, except from within the hypervisor.

The following table lists the registers that are used to control access to disk images resident on the SD card:

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D68A	54922	D1D64	D0D64	-					
D68B	54923	D1MD	D0MD	D1WP	D1P	D1IMG	D0WP	D0P	D0IMG
D68C	54924	D0STARTSEC0							
D68D	54925	D0STARTSEC1							
D68E	54926	D0STARTSEC2							
D68F	54927	D0STARTSEC3							
D690	54928	D1STARTSEC0							
D691	54929	D1STARTSEC1							
D692	54930	D1STARTSEC2							
D693	54931	D1STARTSEC3							
D6A1	54945	-				SILENT	USE-REAL1	TARGANY	USE-REAL0

- **D0D64** F011 drive 0 disk image is D64 mega image if set (otherwise 800KiB 1581 or D65 image)
- **D0IMG** F011 drive 0 use disk image if set, otherwise use real floppy drive.
- **D0MD** F011 drive 0 disk image is D65 image if set (otherwise 800KiB 1581 image)
- **D0P** F011 drive 0 media present
- **D0STARTSEC0** F011 drive 0 disk image address on SD card (LSB)
- **D0STARTSEC1** F011 drive 0 disk image address on SD card (2nd byte)
- **D0STARTSEC2** F011 drive 0 disk image address on SD card (3rd byte)
- **D0STARTSEC3** F011 drive 0 disk image address on SD card (MSB)
- **D0WP** Write enable F011 drive 0

- **D1D64** F011 drive 1 disk image is D64 image if set (otherwise 800KiB 1581 or D65 image)
- **D1IMG** F011 drive 1 use disk image if set, otherwise use real floppy drive.
- **D1MD** F011 drive 1 disk image is D65 image if set (otherwise 800KiB 1581 image)
- **D1P** F011 drive 1 media present
- **D1STARTSEC0** F011 drive 1 disk image address on SD card (LSB)
- **D1STARTSEC1** F011 drive 1 disk image address on SD card (2nd byte)
- **D1STARTSEC2** F011 drive 1 disk image address on SD card (3rd byte)
- **D1STARTSEC3** F011 drive 1 disk image address on SD card (MSB)
- **D1WP** Write enable F011 drive 1
- **SILENT** Disable floppy spinning and tracking for SD card operations.
- **TARGANY** Read next sector under head if set, ignoring the requested side, track and sector number.
- **USEREAL0** Use real floppy drive for drive 0 if set (read-only, except for from hypervisor)
- **USEREAL1** Use real floppy drive for drive 1 if set (read-only, except for from hypervisor)

F011 Virtualisation

In addition to allowing automatic read and write access to SD card based D81 images, it is possible to connect a program to the serial monitor interface that provides and accepts data as though it were the floppy disk.

This is commonly used in a cross-development environment, where you wish to frequently modify a disk image that is used by a program you are developing – without the need to continually push new versions of the disk image on the MEGA65's SD card first. It also has the added benefit that it allows you to easily visualise which sectors are being read from and written to, which can help speed up development and debugging of your program.

This function operates together with the MEGA65's Hypervisor by triggering hyperrupts (that is, interrupts that activate the Hypervisor). There is then special code in the Hypervisor that communicates with the `m65` program via the serial monitor interface.

If that all sounds rather complex, all you need to know is that to use this function, you run the `m65` utility with arguments like `-d image.d81`. This should automatically establish the link with the MEGA65. If the BASIC interprettor stops responding, press the reset button (not the power switch) on the left-hand side of your MEGA65, and it should return to the BASIC's `READY.` prompt – and if your supplied disk image has a C65 auto-boot function, then it should automatically start booting.

This function works very well if the host computer runs Linux, and will allow loading at a speed of around 60KB per second. However, it may be much slower on Windows or Apple OSX-based systems.

Of course to use this, you will also need an interface module and/or cable to connect your cross-development system to the MEGA65's serial monitor interface. This is most easily done using a Trenz TE0790-03 JTAG adapter and mini-USB cable.

More information on using this interface and the `m65` tool can be found in *the MEGA65 Book*, Data Transfer and Debugging Tools (chapter 16).

Dual-Bus SD card Controller

The 45IO27 contains a high-speed dual-bus SD card controller. This controller operates in SPI x1 mode at a clock speed of 20MHz, providing a maximum throughput of approximately 2MB/sec. The quality of the SD card makes a signficant difference to performance, with some cards routinely delivering 1.7MB/sec, while others 1MB/sec or less. Generally speaking, newer cards marketted as being suitable for video recording perform better. The controller supports SDHC cards, and has experimental support for SDXC cards. Legacy SD cards with a capacity of 2GB or less are not supported, as these use a different addressing mode.

The SD controller itself is very simple to drive: Supply the sector number in $D861-$D684, and then issue a read or write command to the command register ($D680). The SD controller supports only sector-based buffered operations, using the sector buffer. In hypervisor mode, the sector buffer is located at $FFD6E00 - $FFD6FFF, while when the computer is in normal operating mode, the SD card and the floppy controller share a single address for both the floppy drive and SD card sector buffers. Which buffer is visible at that address is dictated by the BUFSEL signal. If it is 1, then the SD card buffer is visible, while if it is 0, then the floppy drive sector buffer is visible. See also Sub-section 8 on page 120 for further discussion on the precise behaviour of this buffer with regard to normal mode versus Hypervisor mode, and how it can also be mapped at $DE00.

Write Gate

When writing a sector, you must, however, first open the "write gate". This is a mechanism to prevent accidental corruption of data on the SD card, as it requires two different values to be written to the command register ($D680) in quick succession: You have approximately 1 milli second after opening the write gate to command the write, before the write gate effectively closes again, write-protecting the SD card until the write gate is opened again. There are two different write gates: One for the master boot record (sector 0), and the other for all other sectors, both of which are listed in the command table below. This is designed to provide additional protection to the very important master boot record sector against programs accidentally calculating sector 0 as the target for an ordinary write.

Fill Mode

Where you wish to fill sectors with a constant value, the 45IO27 supports a mode for this, so that you do not need to overwrite the contents of the sector buffer. This is activated by placing the desired fill value into the FILLVAL register ($D686), and then issuing the enable fill mode command ($83), performing the sector write operations, and then issuing the disable fill mode command ($84).

Selecting Among Multiple SD cards

The controller supports two SD card interfaces, and it is possible to have a card in both at the same time. However, each card needs to be reset and commanded separately. Only one card can be commanded at a time. That said, it is possible to reset each card once, and then switch between the cards to perform individual operations.

To select the first SD card slot, write $C0 to the SD Controller Command Register ($D680). To select the second SD card slot, write $C1 instead.

SD Controller Command Table

The SD controller supports the following commands that can be written to the command register at $D680:

Command	Function
$00 (0)	Place SD card under reset (deprecated. Use command $10 instead)
$01 (1)	Release SD card from reset
$02 (2)	Read a sector from the SD card
$03 (3)	Write a single sector to the SD card
$04 (4)	Write the first sector of a multi-sector write to the SD card
$05 (5)	Write a subsequent sector of a multi-sector write to the SD card
$06 (6)	Write the final sector of a multi-sector write to the SD card
$0C (12)	Request flush of SD card write buffers (experimental)
$0E (14)	Pull SD handshake line low (debug only)
$0F (15)	Pull SD handshake line high (debug only)
$10 (16)	Place SD card under reset with flags set (preferred method)
$11 (17)	Release SD card from reset (alternate method)
$40 (64)	Clear the SDHC/SDXC flag, selecting legacy SD card mode (deprecated)
$41 (65)	Set the SDHC/SDXC mode flag
$44 (68)	End force clearing of SD card state machine error flag

continued ...

...continued

Command	Function
$45 (69)	Begin force clearing of SD card state machine error flag
$4D (77)	Open write-gate to sector 0 (master boot record) for approximately 1 milli-second
$57 (87)	Open write-gate for all sectors > 0 for approximately 1 milli-second
$81 (129)	Enable mapping of the SD/FDC sector buffer at $DE00 - $DFFF
$82 (130)	Disable mapping of the SD/FDC sector buffer at $DE00 - $DFFF
$83 (131)	Enable SD card Fill Mode
$84 (132)	Disable SD card Fill Mode
$C0 (192)	Select SD card Slot 0
$C1 (193)	Select SD card Slot 1

Note that the hypervisor can enable or disable direct access to the SD controller. The hypervisor operating system may provide a mechanism for requesting permission to access the SD card controller, e.g., for disk management utilities.

The SD card controller registers are as follows:

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D680	54912	CMDANDSTAT							
D681	54913	SECTOR0							
D682	54914	SECTOR1							
D683	54915	SECTOR2							
D684	54916	SECTOR3							
D686	54918	FILLVAL							
D68A	54922	-				VFDC1	VFDC0	VICIII	CDC00
D6AE	54958	FDCTIBEN	FDC-2XSEL	FDC-VARSPD	AUTO-2XSEL	FDCENC			
D6AF	54959	-		VLOST	VDRQ	VRNF	VEQINH	VW-FOUND	VRFOUND

- **AUTO2XSEL** Automatically select DD or HD decoder for last sector display
- **CDC00** (read only) Set if colour RAM at $DC00
- **CMDANDSTAT** SD controller status/command
- **FDC2XSEL** Select HD decoder for last sector display
- **FDCENC** Select floppy encoding (0=MFM, 1=RLL2,7, F=Raw encoding)
- **FDCTIBEN** Enable use of Track Info Block settings

- **FDCVARSPD** Enable automatic variable speed selection for floppy controller using Track Information Blocks on MEGA65 HD floppies
- **FILLVAL** WRITE ONLY set fill byte for use in fill mode, instead of SD buffer data
- **SECTOR0** SD controller SD sector address (LSB)
- **SECTOR1** SD controller SD sector address (2nd byte)
- **SECTOR2** SD controller SD sector address (3rd byte)
- **SECTOR3** SD controller SD sector address (MSB)
- **VDRQ** Manually set f011_drq signal (indented for virtual F011 mode only)
- **VEQINH** Manually set f011_eq_inhibit signal (indented for virtual F011 mode only)
- **VFDC0** (read only) Set if drive 0 is virtualised (sectors delivered via serial monitor interface)
- **VFDC1** (read only) Set if drive 1 is virtualised (sectors delivered via serial monitor interface)
- **VICIII** (read only) Set if VIC-IV or ethernet IO bank visible
- **VLOST** Manually set f011_lost signal (indented for virtual F011 mode only)
- **VRFOUND** Manually set f011_rsector_found signal (indented for virtual F011 mode only)
- **VRNF** Manually set f011_rnf signal (indented for virtual F011 mode only)
- **VWFOUND** Manually set f011_wsector_found signal (indented for virtual F011 mode only)

TOUCH PANEL INTERFACE

Some MEGA65 variants include an LCD touch panel, primarily the MEGAphone handheld version of the MEGA65. The touch interface supports the detection of two simultaneous touch events. Some variants may also support gesture detection, however, this is still very experimental.

The touch detection interface that is contained in the 45IO27 is complemented by the on-screen-keyboard interface of the 4551 UART and GPIO controller. Refer to section 6 for further information. Of particular relevance are bit 7 of the registers $D615 - $D617 which allow activating the on-screen keyboard interface, selecting whether the on-screen keyboard is placed in the upper or lower portion of the screen, and whether the primary or secondary on-screen keyboard is displayed.

Direct connections between the 4551 and the 45IO27 combine information about any currently displayed on-screen keyboard and the touch interface controller, al-

lowing synthetic keyboard events to be automatically triggered when the on-screen keyboard portion of the touch interface is pressed. This allows the touch interface to be used to drive the on-screen keyboard without requiring any support from user programs. This works even when the on-screen keyboard is moving during activation or transitioning between the top and bottom of the screen.

As touch interfaces can require calibration, the 45IO27 allows for a linear transformation of both the X and Y coordinates of a touch event. Specifically, there are scale (TCHXSCALE and TCHYSCALE) and offset registers (TCHXDELTA and TCHYDELTA) that provide for this transformation. It is also possible to flip the touch screen coordinates in either or both the X and Y axes. These calibration registers also affect the operation of the on-screen keyboard.

It should also be noted that some touch interfaces do not have constant horizontal or vertical resolution. For example, some panels have a low horizontal resolution region in the middle of the panel, which can require some care to accommodate.

To detect the primary touch event, the TOUCH1XLSB, TOUCH1XMSB, TOUCH1YLSB, TOUCH1YMSB registers can be read. Similar registers exist for the 2nd touch event: TOUCH2XLSB, TOUCH2XMSB, TOUCH2YLSB, TOUCH2YMSB. Each touch event has a signle bit flag that indicates whether the touch event is currently valid: the EV1 and EV2 bits of the register $D6B0. There are also corresponding bit-fields that indicate whether a given touch event has been made or released, allowing the detection of when a finger both makes and breaks contact with the screen. The UPDN1 and UPDN2 signals provide this information. Binary values of 01 and 10, respectively indicate if the finger has been removed or pressed against the touch panel. Values of 00 and 11 mean that a finger is either being held or not being held against the touch panel.

The primary touch event is also fed into the lightpen input of the VIC-IV, and can be detected using the normal light pen registers of the VIC-IV.

The registers for the touch panel interface are as follows:

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D6B0	54960	YINV	XINV	UPDN2		UPDN1		EV2	EV1
D6B1	54961	CALXSCALELSB							
D6B2	54962	CALXSCALEMSB							
D6B3	54963	CALYSCALELSB							
D6B4	54964	CALYSCALEMSB							
D6B5	54965	CALXDELTALSB							
D6B7	54967	CALYDELTALSB							
D6B8	54968	CALYDELTAMSB							
D6B9	54969	TOUCH1XLSB							
D6BA	54970	TOUCH1YLSB							
D6BB	54971	-		TOUCH1YMSB		-		TOUCH1XMSB	
D6BC	54972	TOUCH2XLSB							
D6BD	54973	TOUCH2YLSB							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D6BE	54974	-		TOUCH2YMSB		-		TOUCH2XMSB	
D6C0	54976	GESTUREID				GESTUREDIR			

- **CALXDELTALSB** Touch pad X delta LSB
- **CALXSCALELSB** Touch pad X scaling LSB
- **CALXSCALEMSB** Touch pad X scaling MSB
- **CALYDELTALSB** Touch pad Y delta LSB
- **CALYDELTAMSB** Touch pad Y delta MSB
- **CALYSCALELSB** Touch pad Y scaling LSB
- **CALYSCALEMSB** Touch pad Y scaling MSB
- **EV1** Touch event 1 is valid
- **EV2** Touch event 2 is valid
- **GESTUREDIR** Touch pad gesture directions (left,right,up,down)
- **GESTUREID** Touch pad gesture ID
- **TOUCH1XLSB** Touch pad touch #1 X LSB
- **TOUCH1XMSB** Touch pad touch #1 X MSBs
- **TOUCH1YLSB** Touch pad touch #1 Y LSB
- **TOUCH1YMSB** Touch pad touch #1 Y MSBs
- **TOUCH2XLSB** Touch pad touch #2 X LSB
- **TOUCH2XMSB** Touch pad touch #2 X MSBs
- **TOUCH2YLSB** Touch pad touch #2 Y LSB
- **TOUCH2YMSB** Touch pad touch #2 Y MSBs
- **UPDN1** Touch event 1 up/down state
- **UPDN2** Touch event 2 up/down state
- **XINV** Invert horizontal axis
- **YINV** Invert vertical axis

AUDIO SUPPORT FUNCTIONS

The 45IO27 provides the primary interface into the MEGA65's full cross-bar audio mixer. This includes the interface for reading or modifying the mixer co-efficients, as

well as accessing the mixer feedback registers, and setting the 16-bit digital sample values that are two of the input channels into the audio mixer.

The audio mixer consists of 128 coefficients, each of which is 16 bits. Each audio output channel, e.g., left speaker, right speaker, left headphone, right headphone, cellular modem 1 (MEGAphone models only) and so on, are generated by taking each of the audio input channels, multiplying them by the appropriate coefficient, and adding it to the total output of the audio output channel.

Because each audio output channel has its own set of coefficients that are applied to all of the audio input channels, this means that it is possible to produce totally different audio out each audio channel: For example, it is possible to play your favourite quadrophonic SID music out of the headphones while rick-rolling passers by with Amiga-style MOD audio. This is why the audio mixer is refered to as a `full cross-bar` mixer, because there are no restrictions on how you mix each audio output channel. In this regard, it is very similar to a full-function audio desk, allowing different mixing levels for different speakers.

Because the audio coefficients are 16 bits each, each one is formed using two successive bytes of the audio co-efficient space. Changes to the audio coefficients take effect immediately, so care should be taken when changing coefficients to avoid audible clicks and pops. Also, you must allow 32 cycles to elapse before changing the selected audio coefficient, as otherwise the change may be discarded if the audio mixer accumulator has not had time to re-visit that coefficient.

The audio sources on the MEGA65 and MEGAphone devices are as follows:

Input Channel ID	**Connection**
$0 (0)	Left SIDs
$1 (1)	Right SIDs
$2 (2)	Modem Bay 1 (MEGAphone only)
$3 (3)	Modem Bay 2 (MEGAphone only)
$4 (4)	Bluetooth™ Left
$5 (5)	Bluetooth™ Right
$6 (6)	Headphone Interface 1
$7 (7)	Headphone Interface 2
$8 (8)	Digital audio Left
$9 (9)	Digital audio Right
$A (10)	MEMs Microphone 0 (Nexys4 and MEGAphone only)
$B (11)	MEMs Microphone 1 (MEGAphone only)
$C (12)	MEMs Microphone 2 (MEGAphone only)
$D (13)	MEMs Microphone 3 (MEGAphone only)
$E (14)	Headphone jack microphone (Nexys4 and MEGAphone only)
$F (15)	OPL-compatible FM audio (*shares co-efficient with input 14*)

The OPL-compatible FM audio which is on source 15 is controlled by the coefficient for source 14. This is because the coefficient for source 15 provides the master volume level for each output. The OPL-compatible FM audio device is not currently functional in the MEGA65 core.

The audio cross-bar mixer supports the following eight output channels:

Output Channel ID	Connection
$0 (0)	Left Primary Speaker (digital audio on MEGA65 R2/R3, physical speaker on MEGAphone, headphone jack audio on Nexys4)
$1 (1)	Right Primary Speaker (digital audio on MEGA65 R2/R3, physical speaker on MEGAphone, headphone jack audio on Nexys4)
$2 (2)	Modem Bay 1 audio output (MEGAphone only)
$3 (3)	Modem Bay 2 audio output (MEGAphone only)
$4 (4)	Bluetooth Left Audio (MEGAphone only)
$5 (5)	Bluetooth Right Audio (MEGAphone only)
$6 (6)	Headphone Left output (MEGA65 R2/R3 and MEGAphone only. On Nexys4 boards the primary speaker drives the 3.5mm jack)
$7 (7)	Headphone Right output (MEGA65 R2/R3 and MEGAphone only. On Nexys4 boards the primary speaker drives the 3.5mm jack)

To determine the coefficient register number for a given source and output, multiply the output number by 32 and multiply the source number by 2. This will be the register number for the LSB of the 16-bit coefficient. The MSB will be the next register. For example, to set the coefficient of the right SIDs to the 2nd modem bay audio output, the coefficient would be $32 \times 3 + 1 \times 2 = 96 + 2 = 98$.

Other Audio Features

The audio sub-system supports several other features, that are currently pending further documentation.

Mixer Feedback Registers

These registers allow the processor to access the mixed audio for a particular output channel. This can be used to implement displays of audio waveforms, or implement certain audio-effects, such as reverb.

8/16 Bit Stereo Digital Audio Registers

Registers are provided for injecting 8 or 16 bit audio samples directly into dedicated input channels of the audio mixer, providing a simple way to play digital audio data. This is particularly useful for procedurally generated audio data. For recorded samples, it is generally simpler and better to use the Audio DMA functionality that fully automates the play-back of digital audio. Audio DMA is especially heplful at higher sample rates, as it reduces the burden on the CPU, and greatly reduces sample jitter.

Pulse Width vs Pulse Density Modulation

For models of the MEGA65 that use a 1-bit over-sampled output for audio (upto and including the R3/R3A model), it is possible to select between these two different over-sampling methods. Both have similar performance, but users may prefer one over the other. This choice has no effect on the R4 or newer models that use a dedicated audio DAC to feed audio to the 3.5mm audio jack, and that has intrinsically better audio quality. For owners of older models, the planned internal expansion board for the MEGA65 may include an improved audio output circuit, depending on the final configuration of that board. No timeline is currently available for the availability of this board. In the interim, use of HDMI audio output is the recommended solution, as the audio is encoded digitally over the HDMI cable.

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D6F4	55028	MIXREGSEL							
D6F5	55029	MIXREGDATA							
D6F8	55032	DIGILLSB							
D6F9	55033	DIGILMSB							
D6FA	55034	DIGIRLSB							
D6FB	55035	DIGIRMSB							
D6FC	55036	READBACKLSB							
D6FD	55037	READBACKMSB							
D711	55057	-				PWMPDM	-		

- **DIGILEFTLSB** Digital audio, left channel, LSB
- **DIGILEFTMSB** Digital audio, left channel, MSB
- **DIGILLSB** 16-bit digital audio out (left LSB)
- **DIGILMSB** 16-bit digital audio out (left MSB)
- **DIGIRIGHTLSB** Digital audio, left channel, LSB
- **DIGIRIGHTMSB** Digital audio, left channel, MSB
- **DIGIRLSB** 16-bit digital audio out (right LSB)
- **DIGIRMSB** 16-bit digital audio out (right MSB)
- **MIXREGDATA** Audio Mixer register read port

- **MIXREGSEL** Audio Mixer register select
- **PWMPDM** PWM/PDM audio encoding select
- **READBACKLSB** audio read-back LSB (source selected by $D6F4)
- **READBACKMSB** audio read-back MSB (source selected by $D6F4)

MISCELLANEOUS I/O FUNCTIONS

CHAPTER 9

4541 Serial Bus Controller

- Overview
- Features of the 4541
- Theory of Operation
- Examples
- Command Reference
- Register Table
- Serial Bus Timing
- Optional Integrated Data-Logger

OVERVIEW

The 4541 is a Commodore™ serial peripheral bus compatible bus controller, that greatly reduces the effort required to communicate with devices on this bus.

FEATURES OF THE 4541

Supports Enhanced Serial Protocol Variants

The 4541 supports the JiffyDOS™ extensions to this protocol, that allow data transfers approximately 10× faster than using the original protocol. It is also expected that a future revision of the 4541 will support Commodore's fast serial protocol, that is present in the 1571 and 1581 disk drives.

Interrupt Enabled Processor Offload

The 4541 performs serial communications independent of the microprocessor. Together with an IRQ functionality, this allows software to continue on other tasks while serial peripheral communications occurs, requiring only to be briefly interrupted when the next event on the serial peripheral bus occurs.

Processor Speed Independence

A major advantage of the 4541, is that it also handles all timing requirements of communications on this bus, allowing the bus to be driven by a processor that can run at different speeds, without having to modify the the bus controller software.

Co-Existence through Open-Collector Logic

Because the 4541 uses open-collector logic, it can be used in parallel with existing software-based implementations of the bus protocol, ensuring compatibility with existing software. It is installed in this configuration in the MEGA65, allowing the legacy software-based serial peripheral communications software that controls the serial peripheral bus to continue to be used unmodified.

THEORY OF OPERATION

The 4541 presents a quite simple interface: You issue commands, wait for a response, and retrieve any data that the command retrieved. Some commands also require a data byte, which is provided by a dedicated register. There is also a device info register, that lets you see what the 4541 believes about the current status of the most re-

cently requested device, including whether it is present, and whether it supports either or both of the JiffyDOS™ or Commodore™ 128 extensions to the standard protocol.

So, for example, to release the Attention line, you can simply write the appropriate command byte value (65 = $41) to the command register at $D698, and then check for the completion status in the status register at $D697, as shown in the following example written in BASIC65:

```
10 POKE $D698,$41
20 IF ( PEEK($D697) AND $20 ) = $00 GOTO 20
30 PRINT "DONE"
```

The 4541 implements a set of commands that map very closely to the KERNAL calls that are used to control the IEC bus on the C64 and related computers. In most cases, there is a single corresponding command for the 4541, although in a few cases, you may need to issue two commands, as summarised in the following table:

KERNAL Call	Meaning and Equivalent 4541 Command(s)
$FF93 LSTNSA	Send LISTEN secondary address. 4541 Equivalent: Data = Binary OR of $60 and the desired secondary address. Command $30. Then Command $41 to release the Attention line.
$FF96 TALKSA	Sent TALK secondary address. 4541 Equivalent: Data = Binary OR of $60 and the desired secondary address. Command $30. Then Command $41 to release the Attention line.
$FFA5 IECIN	Receive a byte from the serial peripheral bus. 4541 Equivalent : Command $32. Received byte is available in the data register on completion.
$FFA8 IECOUT	Send a byte to the serial peripheral bus. 4541 Equivalent : Data = the byte to send. Command $31 (or $30 if the byte is to be sent under Attention).
$FFA8 UNTALK	Send UNTALK command to serial peripheral bus. 4541 Equivalent : Data = $5F. Command $30.
$FFAB UNLISTN	Send UNLISTEN command to serial peripheral bus. 4541 Equivalent : Data = $3F. Command $30.
$FFB1 LISTEN	Send LISTEN command to the serial peripheral bus.

KERNAL Call	Meaning and Equivalent 4541 Command(s)
	4541 Equivalent : Data = $20 plus the device number. Command $30.
$FFB4 TALK	Send TALK command to the serial peripheral bus. 4541 Equivalent : Data = $40 plus the device number. Command $30.
$FFB7 READST	Read the status of the serial peripheral bus. 4541 Equivalent : Read the status bits from $D698. For convenience, the upper bits of this status byte have the same layout as used in the KERNAL.

The 4541 is very obedient: If you ask it to do something new, it will start doing that immediately, even if it was in the middle of doing something else – even if it was halfway through sending a byte of data to the serial peripheral bus!

You should therefore always wait until the IRQREADY bit in $D697 is set before issuing each command, or reading the status bits, to make sure that the controller has finished whatever it was last asked to do.

EXAMPLES

Reading the DOS channel status

The following BASIC65 program can be used to talk to a device connected to the serial peripheral bus. Note that because it uses the 4541, and not the C65/MEGA65 CBDOS, it ignores the presence of the internal 3.5" disk drive, because that drive is not connected to the serial peripheral bus.

```
10 REM ABORT ANY EXISTING 4541 COMMAND IN PROGRESS
20 POKE $D698,$00
30 REM RESET THE SERIAL PERIPHERAL BUS BY PULLING
40 REM THE RESET PIN LOW FOR 1 SECOND
50 POKE $D698,$72 : SLEEP 1 : POKE $D698,$52
60 REM GIVE CONNECTED DEVICES TIME TO GET READY AFTER RESET
70 SLEEP 2: REM THE 1541 TAKES QUITE A WHILE TO RESET!
80 REM SELECT THE DEVICE NUMBER TO TALK TO
90 D = 8
100 PRINT"DEVICE"; D; "TALK"
110 POKE $D699,$40 + D : POKE $D698,$30 : GOSUB 1000
120 PRINT "SECONDARY ADDRESS 15"
130 POKE $D699,$6F : POKE $D698,$30 : GOSUB 1000
140 PRINT"TURN AROUND TO LISTEN"
150 POKE $D698,$35 : GOSUB 1000
160 PRINT "READ DOS STATUS"
170 POKE $D698,$32 : GOSUB 1000
180 PRINT CHR$( PEEK($D699) );
190 CC = CC + 1 : IF CC < 200 GOTO 180
999 END
1000 REM WAIT FOR 4541 TO FINISH COMMAND
1010 S=PEEK($D697) : IF (S AND 32) = 32 GOTO 1020
1020 GOTO 1000
1030 IF S AND 128 THEN PRINT "DEVICE NOT PRESENT" : END
1040 RETURN
```

If you have a 1581 with a JiffyDOS™ ROM connected, this program will display something like the following when run:

```
READY.
RUN
DEVICE 8 TALK
SECONDARY ADDRESS 15
TURN AROUND TO LISTEN
READ DOS STATUS
73,(C) 1989 JIFFYDOS 6.0 1581,00,00
00, OK,00,00
00, OK,00,00
00, OK,00,00
...
00, OK,0
READY.
```

COMMAND REFERENCE

The following table lists the set of command codes suppoprted by the 4541.

Command Byte	Action
Abort Running Commands	
$00	Abort any executing command.
$01	Reset controller state: Release ATN, CLK, DATA, SRQ and enable default protocol selection.
Bit-Bashing Commands	
$41	Release Attention (ATN) line to 5V.
$61	Pull Attention (ATN) line to 0V.
$43	Release clock (CLK) line to 5V.
$63	Pull clock (CLK) line to 0V.
$44	Release data (DATA) line to 5V.
$64	Pull data (DATA) line to 0V.
$53	Release fast serial clock (SRQ) line to 5V.
$73	Pull fast serial clock (SRQ) line to 0V.
$52	Pull reset (RESET) line to 0V.
$72	Release reset (RESET) line to 5V.
Protocol Variant Control Commands	
$4A	Enable solicitation of JiffyDOS™ extension to the serial protocol.
$6A	Disable solicitation of JiffyDOS™ extension to the serial protocol.
$46	Enable solicitation of Commodore™ 1571/1581 fast serial extension to the serial protocol.
$66	Disable solicitation of Commodore™ 1571/1581 fast serial extension to the serial protocol.
Serial Bus Protocol Commands	
$30	Send byte in $D699 under attention. Attention line is held at 0V at the end of the transaction.

Command Byte	Action
$31	Send bye in $D699 without attention. The attention line is released, if it was previously asserted.
$32	Receive a byte from the peripheral serial bus. There must be a previously activated talker. The received byte is stored in $D699.
$33	RESERVED
$34	Send a byte and indicate EOI. Attention is released prior to transmission, if was asserted.
$35	Turn around from talker to listener.

Protocol Timing Commands

$80	Reset protocol timing to defaults.
$81	Set T_R delay (1 - 255 microseconds). Old value can be read from $D699.
$82	Set T_{TK} delay (1 - 255 microseconds). Old value can be read from $D699.
$83	Set T_{DC} timeout (1 - 255 milliseconds). Old value can be read from $D699.
$84	Set T_{BB} delay (1 - 255 microseconds). Old value can be read from $D699.
$85	Set T_{HA} delay (1 - 255 microseconds). Old value can be read from $D699.
$86	Set T_{ST} delay (1 - 255 microseconds). Old value can be read from $D699.
$87	Set T_{VT} delay (1 - 255 microseconds). Old value can be read from $D699.
$88	Set T_{AL} delay (1 - 255 microseconds). Old value can be read from $D699.
$89	Set T_{AC} delay (1 - 255 microseconds). Old value can be read from $D699.
$8A	Set T_{AT} delay (1 - 255 milliseconds). Old value can be read from $D699.

Command Byte	Action
$8B	Set T_H delay (1 – 255 milliseconds). Old value can be read from $D699. *This value currently has no effect, as the 4541 allows truly infinite data-hold off by a listener, except for bytes sent under attention, where T_{HA} is used instead.*
$8C	Set T_{NE} delay (1 – 255 microseconds). Old value can be read from $D699.
$8D	Set T_F delay (4 – 1020 microseconds). Old value can be read from $D699, scaled by 4.
$8E	Set T_{YE} delay (1 – 255 microseconds). Old value can be read from $D699.
$8F	Set T_{EI} delay (1 – 255 microseconds). Old value can be read from $D699.
$90	Set T_{AR} delay (1 – 255 microseconds). Old value can be read from $D699.
$91	Set T_{JT} delay (4 – 1020 microseconds). Old value can be read from $D699, scaled by 4.
$92	Set T_{J0} delay (1 – 255 microseconds). Old value can be read from $D699.
$93	Set T_{J1} delay (1 – 255 microseconds). Old value can be read from $D699.
$94	Set T_{J2} delay (1 – 255 microseconds). Old value can be read from $D699.
$95	Set T_{J3} delay (1 – 255 microseconds). Old value can be read from $D699.
$96	Set T_{J4} delay (1 – 255 microseconds). Old value can be read from $D699.
$97	Set T_{J5} delay (1 – 255 microseconds). Old value can be read from $D699.
$98	Set T_{J6} delay (1 – 255 microseconds). Old value can be read from $D699.
$99	Set T_{J7} delay (1 – 255 microseconds). Old value can be read from $D699.
$9A	Set T_{J8} delay (1 – 255 microseconds). Old value can be read from $D699.

Command Byte	Action
$9B	Set T_{J9} delay (1 - 255 microseconds). Old value can be read from $D699.
$9C	Set T_{J10} delay (1 - 255 microseconds). Old value can be read from $D699.
$9D	Set T_{J11} delay (1 - 255 microseconds). Old value can be read from $D699.
$9E	Set T_{JR} delay (1 - 255 microseconds). Old value can be read from $D699.
$9F	Set T_{FS} delay (1 - 255 microseconds). Old value can be read from $D699.
$A0	Set T_{FF} delay (1 - 255 microseconds). Old value can be read from $D699.
$A1	Set T_{PULLUP} delay (1 - 255 microseconds). Old value can be read from $D699.
$A2	Set T_{JD} delay (4 - 1020 microseconds). Old value can be read from $D699, scaled by 4.
$A3	Set T_{J12} delay (1 - 255 microseconds). Old value can be read from $D699.

Diagnostic Commands

$D0	Trigger optional integrated data logger, if enabled.
$D1	Begin transmitting a 1KHz pulse train on the CLK and DATA lines. Continues until aborted by another command.

REGISTER TABLE

The 4541 has the following registers:

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D694	54932	DATALOG0							
D695	54933	DATALOG1							
D697	54935	IRQFLAG	IRQRX	IRQRDY	IRQTO	IRQEN	IRQRXEN	IRQRDYEN	IRQTOEN
D698	54936	STNODEV	STNOEOI	STSRQ	STVERIFY	STC	STD	STTO	STDDIR

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D699	54937	DATA							
D69A	54938	PRESENT	PROT		DIATN	DEVNUM			

- **DATA** Data byte read from IEC bus
- **DATALOG0** Access integrated data logger in IEC controller
- **DATALOG1** Access integrated data logger in IEC controller
- **DEVNUM** Lower 4 bits of currently selected device number
- **DIATN** Device is currently held under attention
- **IRQEN** Enable interrupts if set
- **IRQFLAG** Interrupt flag. Set if any IRQ event is triggered.
- **IRQRDY** Set if ready to process a command
- **IRQRDYEN** Enable TX interrupt source if set
- **IRQRX** Set if a byte has been received from a listener.
- **IRQRXEN** Enable RX interrupt source if set
- **IRQTO** Set if a protocol timeout has occurred, e.g., device not found.
- **IRQTOEN** Enable timeout interrupt source if set
- **PRESENT** Device is present
- **PROT** Device protocol support (5=C128/C65 FAST, bit 6 = JiffyDOS(tm))
- **STC** State of CLK line
- **STD** State of DATA line
- **STDDIR** Data direction when timeout occurred.
- **STNODEV** Device not present
- **STNOEOI** End of Indicate (EOI/EOF)
- **STSRQ** State of SRQ line
- **STTO** Timeout occurred
- **STVERIFY** Verify error occurred

SERIAL BUS TIMING

This section describes the timing requirements primarily from the perspective of a bus controller. If you are designing serial bus peripherals, please take care to carefully understand how these time requirements affect peripherals: The

Send Byte Under Attention

When the controller wishes to get the attention of one or more peripherals on the bus, it uses the attention (ATN) line to indicate this: It pulls it low to 0V, and then sends one or more bytes that will be received by all devices on the bus.

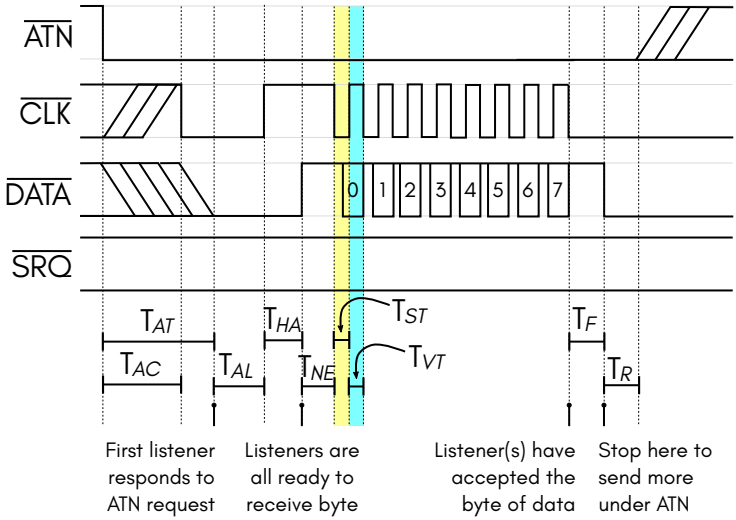
The SRQ line is not active in this transaction.

To summarise:

1. The controller pulls the ATN to 0V and releases the CLK, DATA and SRQ lines if it was holding any of them at 0V.
2. All peripherals on the bus as they notice the ATN line at 0V begin to indicate this by pulling the DATA line to 0V. The controller can't tell how many, or whether all the devices on the bus have responded: Rather, it can only tell that at least one device has.
3. If no device responds, DATA stays at 5V, because no one is pulling it down, and the controller will indicate a DEVICE NOT PRESENT error.
4. While the controller is waiting for devices to respond, it also pulls the CLK line to 0V, typically a short time after pulling the ATN line to 0V.
5. After the controller has waited a while, typically 1 millisecond, it assumes that all peripherals who are going to answer the ATN request have.
6. Next, the controller releasing the CLK line to 5V to indicate that it wants to send a byte of data on the bus. This is the start of the transmission of a byte under ATN. All of the previous steps have had the purpose of establishing the ATN communications.
7. The controller now waits for all devices to indicate their readiness to receive a byte by releasing the DATA line back to 5V. Because the bus is open-collector, if even a single device is not ready, it will be able to keep pulling the DATA line down to 0V, causing the controller to wait. If this takes too long, the controller may give up and indicate a timeout condition.
8. Once the last device has indicate it is ready to receive data, the controller waits a little longer, to make sure that all of the devices are able to do their final preparations to receive a byte. This doesn't take very long.
9. After that delay, the controller begins sending the 8 bits of data, beginning with the least significant bit (LSB), that is bit 0. For each bit, it first brinks the CLK line low to indicate that data is being loaded onto the bus, pulls the DATA line to 0V

if it wants to send a 0 bit, or lets it float to 5V if it wants to send a 1 bit, and then releases the CLK line back to 5V, and holds it there for a while. The timing of this process is critical: If the CLK is low or high for too short a period of time, the peripherals will get confused, and possibly miss one or more bits, resulting in general chaos on the bus.

10. After the controller has sent the last bit, it holds the CLK line low, and releases the DATA line. It expects one or more peripherals to pull the DATA line to 0V within a short time to acknowledge reception of the byte. If this does not occur, the controller may report a timeout or DEVICE NOT PRESENT error.
11. At the conclusion of this, the controller is holding the CLK line at 0V, and the peripheral(s) are holding the DATA line at 0V. This combination serves to tell each other that the controller is not yet wanting to send the next byte, and that the peripheral(s) are not yet ready to receive the next byte.
12. If the controller wishes to send more bytes under attention, it will repeat this process from the step where it released the CLK line to 5V.
13. Alternatively, if it was the last byte to be sent under this attention request, the controller waits a short time, and then releases the ATN line.



Symbol	Min	4541	Max	Description
T_{AT}	–	1000	∞	Device attention response timeout. *A device must respond to the ATN request within this time. If not device responds within this time, a DEVICE NOT PRESENT error will result. Peripherals should therefore conform to the 1,000 microsecond typical value.*
T_{AC}	–	20	∞?	Time between pulling ATN low, before CLK is also pulled low.
T_{AL}	–	1000	∞?	Time between first device responds to ATN and the controller releases CLK to 5V.
T_{HA}	–	64 ms	∞	Listener hold-off. *The protocol allows a listener can hold of for any desired period of time, or for no time at all. The 4541 does not allow this during ATN requests.*
T_{NE}	–	40	200	Non-EOI timing-channel response to Ready For Data. *If this interval is too long, the byte will be interpreted as an EOI byte, and the peripherals will require an EOI response (see T_{EI}).*
T_{ST}	20*	35	∞	Data bit setup time. *Referred to as T_S in the C64 Programmer's Reference Guide.*
T_{VT}	20*	35	∞	Data bit valid hold time. *Referred to as T_V in the C64 Programmer's Reference Guide.*
T_F	–	1000	1000	Frame handshake (acknowledge) timeout.
T_R	20	200	∞	Time between receiving end of frame acknowledgement and releasing the ATN line to 5V.

All time units are in micro seconds, unless otherwise indicated.

* The Commodore™ 64 Programmer's Reference Guide suggests that T_S and T_V each have a minimum duration of 20 micro seconds. Our investigations suggest that when communicating with a 1541, that safe values for T_{ST} and T_{VT} are closer to 70 micro seconds.

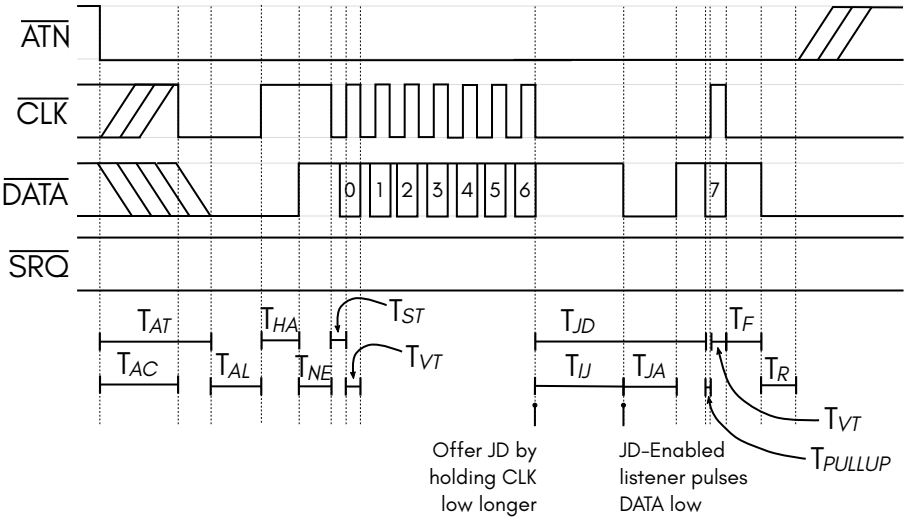
JiffyDOS™ Protocol Solicitation

If support for the JiffyDOS™ protocol extension is enabled, the 4541 will solicit this during the transmission of TALK and LISTEN commands only. That is, when sending a byte under attention that is in the range $20 - $5F.

The SRQ line is not active in this transaction.

To summarise:

1. The transmission of the byte occurs normally, until the T_{ST} period just before the last bit (bit 7) of the byte.
2. Instead of waiting T_{ST}, the controller must instead wait for a much longer period, typically 300 microseconds, so that the listener will recognise that it is being asked if it supports the JiffyDOS™ protocol. The controller also releases the DATA line to 5V.
3. If the listener supports the JiffyDOS™ protocol, it pulls DATA to 0V when it observes that the CLK line has been held low for the longer period.
4. After the listener has held DATA at 0V long enough for it to be sure the controller has had the chance to notice, it releases the DATA line to 5V. The device will now use the JiffyDOS™ for the requested TALK or LISTEN session.
5. The controller notes if the DATA line was pulled to 0V, and if so, records that the listener has selected the JiffyDOS™ protocol, and will use this protocol for the requested TALK or LISTEN session.
6. The controller now sends the final bit of the byte.
7. The controller ensures that the data bit value is visible on DATA before asserting CLK, typically 1 - 4 microseconds.
8. The controller releases the CLK line to 5V to indicate to the listener that the final data bit is ready.
9. The remainder of the byte transfers as normal.



Symbol	Min	4541	Max	Description
T_{JD}	320	320	∞	Time that CLK is held at 0V between the two final bits of the byte to provide the side-channel indication that the controller wishes to use the JiffyDOS™ protocol extension.
T_{IJ}	100	-	295	Time before the listener accepts and begins to acknowledge the selection of the JiffyDOS™ protocol extension.
T_{IJ}	4/100*	-	200	The duration that the listener holds the DATA line low to acknowledge the selection of the JiffyDOS™ protocol extension.

All other timing values are identical to the sending a byte under attention case.

All time units are in micro seconds, unless otherwise indicated.

* The 4541 can detect pulse-widths as narrow as 4 microseconds. However, software implementations of the protocol will require a larger value. JiffyDOS™uses 100 microseconds.

JiffyDOS™ Send from Controller to Peripheral

Unlike the standard serial protocol, the JiffyDOS™ protocol uses different protocols for sending and receiving. This is necessary on software imeplementations to obtain the maximum speed, because the serial peripheral lines are mapped to different register bits, and part of the magic of the JiffyDOS™ protocol is how it cleverly manipulates the received bits to reconstruct the complete byte in the least possible time, and similarly when transmitting.

The result is quite amazing: JiffyDOS™ can send a byte in approximately the same time it takes the original protocol to send a bit!

Note that the JiffyDOS™ protocol transmits data bits with the opposite polarity to the standard protocol, that is a 1 bit is indicated by 0V, while a 0 bit is indicated by 5V.

The SRQ line is not active in this transaction.

In summary,

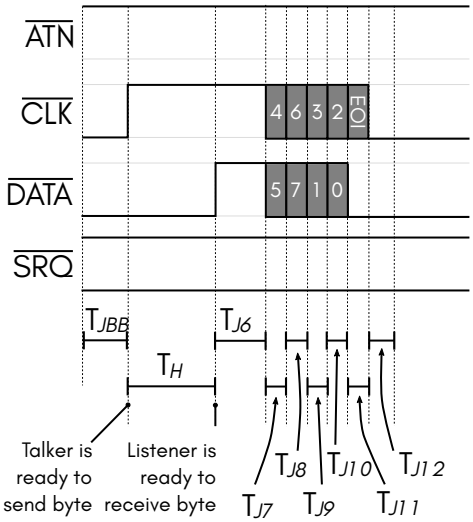
1. Depending on the state of the controller when it begins transmitting, it may be holding the CLK line at 0V. If so it releases it immediately.
2. Next, after some arbitrary time, the peripheral indicates it's readiness to receive a byte from the controller by releasing the DATA line to 5V.
3. The timing for the remainder of the byte transfer is now critical, because immediately following releasing DATA, the peripheral will start reading the CLK and DATA signals to transfer the byte.
4. The controller waits a few microseconds (T_{J6}).
5. The controller sets CLK to 0V if data bit 4 is 1, else releases it to 5V. The controller sets DATA to 0V if data bit 5 is 1, else releases it to 5V.
6. The controller waits a few microseconds (T_{J7}).
7. The controller sets CLK to 0V if data bit 6 is 1, else releases it to 5V. The controller sets DATA to 0V if data bit 7 is 1, else releases it to 5V.
8. The controller waits a few microseconds (T_{J8}).
9. The controller sets CLK to 0V if data bit 3 is 1, else releases it to 5V. The controller sets DATA to 0V if data bit 1 is 1, else releases it to 5V.
10. The controller waits a few microseconds (T_{J9}).
11. The controller sets CLK to 0V if data bit 2 is 1, else releases it to 5V. The controller sets DATA to 0V if data bit 0 is 1, else releases it to 5V.
12. The controller waits a few microseconds (T_{J10}).
13. The controller pulls the DATA line to 0V. It also releases the CLK line to 5V, unless it wishes to indicate EOI, in which case it pulls the CLK line to 0V.

14. The controller waits a few microseconds, during which time the peripheral reads the EOI value, and then pulls DATA to 0V. (T_{J11}).

15. The byte transfer is complete.

Note when sending the first byte under the JiffyDOS™ protocol, that the CLK signal may be at 0V. The time between bytes (T_{JBB}) is not required (set to zero), because the tight receive loop on the peripheral simply indicates when it is ready to receive each next byte, which also contributes to it's high speed.

Note also that the timing below is based on when the signals should be set. For software implementations, the latency of the necessary processor instructions must be taken into account. This is not a problem with the 4541, because it has a latency of less than 50ns.



Symbol	Min	4541	Max	Description
T_{JBB}	0	0	∞	Time between bytes. There is no minimum time between bytes with the JiffyDOS™ protocol, provided T_{J12} is sufficiently large, as that effectively hides the latency of the receive routine.
T_{J6}	10	10	10	JiffyDOS™ transmit receiver setup time.
T_{J7}	13	13	13	JiffyDOS™ transmit hold time, first di-bit.
T_{J8}	11	11	11	JiffyDOS™ transmit hold time, second di-bit.

Symbol	Min	4541	Max	Description
T_{J9}	11	11	11	JiffyDOS™ transmit hold time, third di-bit.
T_{J10}	12	12	12	JiffyDOS™ transmit hold time, fourth di-bit.
T_{J11}	15	15	15	JiffyDOS™ transmit hold time, status bits. CLK carries the EOI notification, and DATA is set to 0V to indicate successful transmission. The receiver may indicate a frame error if DATA is observed to be 5V.
T_{J12}	18	18	18	JiffyDOS™ post transmit recovery time. CLK is pulled to 0V, and DATA is set to 0V to indicate successful transmission. The receiver may indicate a frame error if DATA is observed to be 5V.
T_{JR}	15	15	15	

All other timing values are identical to the sending a byte under attention case.

All time units are in micro seconds, unless otherwise indicated.

JiffyDOS™ Controller Receive from Peripheral

Unlike the standard serial protocol, the JiffyDOS™ protocol uses different protocols for sending and receiving. This is necessary on software imeplementations to obtain the maximum speed, because the serial peripheral lines are mapped to different register bits, and part of the magic of the JiffyDOS™ protocol is how it cleverly manipulates the received bits to reconstruct the complete byte in the least possible time, and similarly when transmitting.

The result is quite amazing: JiffyDOS™ can send a byte in approximately the same time it takes the original protocol to send a bit!

Unlike when transmitting via the JiffyDOS™ protocol where the data bits are inverted, when receiving from a peripheral speaking the JiffyDOS™ protocol, the bits are represented naturally, i.e., with 1 = 5V and 0 = 0V.

The SRQ line is not active in this transaction.

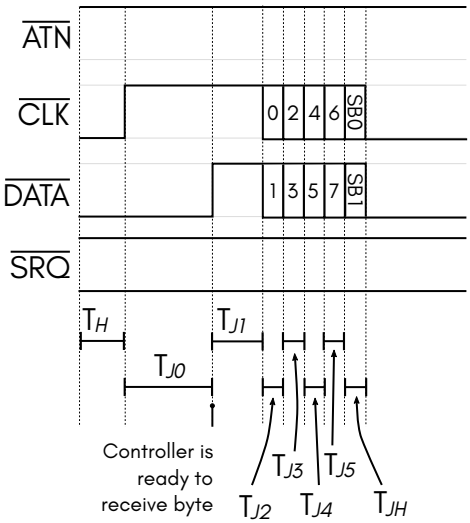
To summarise:

1. The peripheral may wait an arbitrary period of time, before it has a byte to send to the controller. When it does, it releases CLK to 5V.

2. When the controller notices this, it releases DATA to 5V, and begins the critical timing section of the protocol.
3. The peripheral waits a short while. For peripherals using a software-implementation, i.e., all Commodore™ drives, the delay is a natural consequence of the processor instruction involved in commencing sending of the byte, similarly for the following steps.
4. The peripheral places the first di-bit of data on the CLK and DATA lines, and waits a short while.
5. The peripheral places the second di-bit of data on the CLK and DATA lines, and waits a short while.
6. The peripheral places the third di-bit of data on the CLK and DATA lines, and waits a short while.
7. The peripheral places the fourth di-bit of data on the CLK and DATA lines, and waits a short while.
8. The peripheral places the status bits for the byte onto the CLK and DATA lines, and waits a short while.
9. The controller pulls DATA to 0V to acknowledge receipt of the byte, and to prevent the peripheral from sending another byte before it is ready.
10. The peripheral pulls the CLK line to 0V if it does not have another byte ready to send immediately, lease it remains at 5V.

The status bits have the following interpretation:

CLK	DATA	**Meaning**
0V	0V	An error occurred.
0V	5V	Byte received with EOI.
5V	0V	Normal byte received (no EOI).
5V	5V	An error occurred.



Symbol	Min	4541	Max	Description
T_{JH}	10	-	∞	Peripheral hold-off until it has data to send.
T_{J0}	0	-	∞	Controller hold-off until ready to receive a byte.
T_{J1}	37	37	∞	The time is used by software implementations to prepare the byte for immediate transmission.
T_{J2}	14	14	14	Send first di-bit of data
T_{J3}	10	10	10	Send first di-bit of data
T_{J4}	11	11	11	Send first di-bit of data
T_{J5}	11	11	11	Send first di-bit of data
T_{JH}	13	13	13	Send status bits

All other timing values are identical to the sending a byte under attention case.

All time units are in micro seconds, unless otherwise indicated.

Talker to Listener Turn-Around

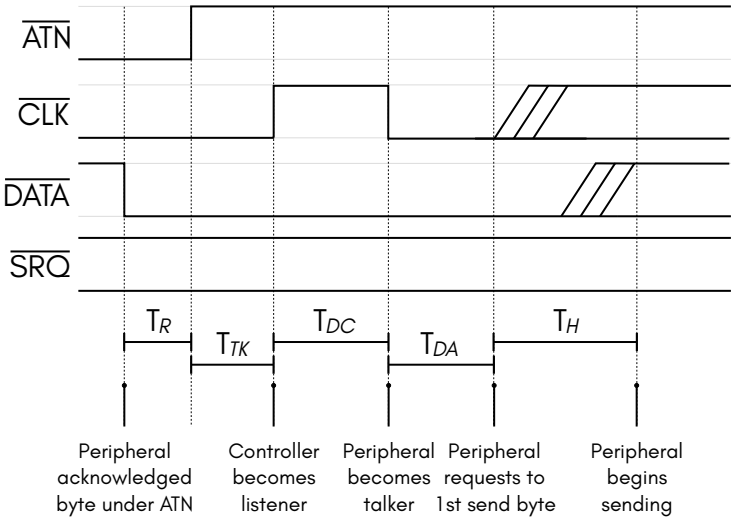
When the bus controller is commanding a device to talk, it finished the transmission with ATN asserted to 0V. The controller is holding the CLK line at 0V, and the device it was talking to is holding the DATA line. This situation needs to be reversed: The controller needs to give up control of the bus, and hand it over to the device.

The SRQ line is not active in this transaction.

To summarise:

1. The controller waits long enough before releasing the ATN line, to make sure that the peripheral(s) has finished acknowledging the byte that was just sent to it under attention. This is the T_R delay.
2. The controller releases the ATN line to 5V.
3. The controller waits a while. This is the T_{TK} delay.
4. The controller releases the CLK line to 5V, and pulls the DATA line to 0V. From this point forward, the controller has relinquished control of the bus, and becomes a listener.
5. After some period of time, the peripheral that has been commanded to talk asserts the CLK line to 0V, and releases the DATA line, which remains at 0V, because the the controller is still pulling it down to 0V in preparation to be the listener. From this point in time, the peripheral has become the sole talker on the bus. The delay before the CLK line is pulled to 0V by the peripheral is the T_{DC} timeout.
6. The peripheral waits a while, before it is allowed to begin talking. This is the T_{DA} delay.

Once this process is complete, the roles of the controller and peripheral have reversed, with the peripheral now having the role of talker, and the controller (and possibly other devices on the bus) of listener.



Symbol	Min	4541	Max	Description
T_{DA}	4/80*	-	∞	Talk-Attention acknowledge hold duration. Controller holds CLK
T_{DC}	0	64 ms	∞	Talk-Attention acknowledge duration, i.e., the time a peripheral is permitted to take before becoming talker. *Can commence immediately when CLK is observed at 5V.*
T_H	-	N/A	∞	Listener hold-off. *Listener can hold of for any desired period of time, or for no time at all.*
T_R	0/20^	200	-	Frame to release of ATN.
T_{TK}	20	40	100	Talk Attention Release.

All time units are in micro seconds, unless otherwise indicated.

* T_{DA} is provided by the peripheral, not the controller. The 4541 is able to respond much faster than a C64 to serial bus events, and thus requires only 4 microsecond to ensure the CLK line has time to rise to 5V. For controllers using software implementations of the protocol, such as the C64, the minimum is 80 microsecond. Therefore peripherals should always use a value of at least 80 microsecond for T_{DA}.

^ The Commodore™ 64 Programmer's Reference Guide lists T_R as having a minimum duration of 20 microsecond. We see no evidence that would suggest that any implementation requires a delay before the ATN line can be released following the transfer of a byte.

Send Byte With End-or-Indicate (EOI)

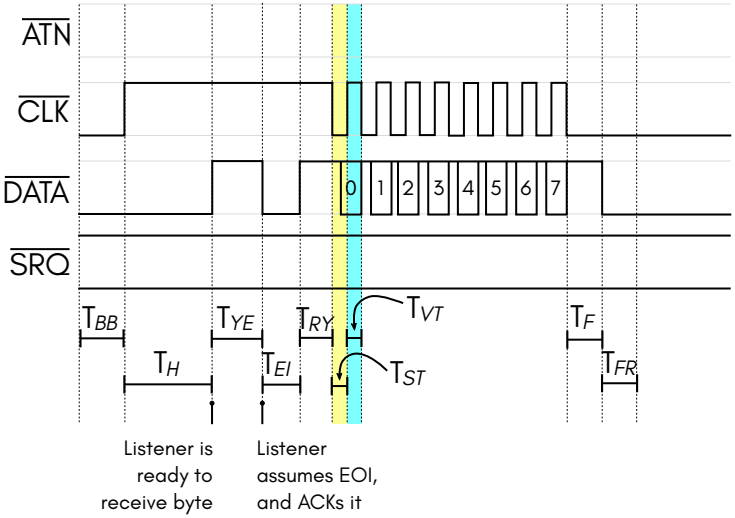
After sending a stream of data, a device will often wish to indicate to the listener that it has reached the end of the data, for example, so that it can begin processing it. This is accomplished by specially marking the byte with the End-or-Indicate (EOI) attribute.

This is implemented using a timing side-channel, so that it does not require a whole dedicated wire. When the talker has floated CLK to 5V to indicate it wishes to send a byte, and when the listener has floated DATA to 5V to indicate that it is ready to receive, the talker can wait at least 200 microseconds, to indicate to the listener that the byte should be received with EOI status. The listener signals this to the talker by pulling the DATA line to 0V for a while, after which the transmission occurs much the same as normal.

The SRQ line is not active in this transaction.

To summarise:

1. The talker releases CLK to 5V to indicate that it wishes to send a byte.
2. After an arbitrary period of time, the listener releases the DATA line to 5V to indicate its readiness to receive a byte.
3. Normally at this point, the talker would begin sending the bits after a short period of time. However, to indicate EOI, it instead waits silently, until the listener gets the idea that this byte is different, resulting in the listener pulling the DATA line back down to 0V.
4. The listener holds the DATA line at 0V for just long enough for it to be sure that the talker has noticed, and then releases it again.
5. As soon as the talker has seen the listener pull the DATA line to 0V and release it again, it begins sending the byte normally after a short delay.
6. If it waits too long, the listener will assume that the talker wanted to indicate EOI without actually sending a byte.
7. The transmission of the byte completes in an identical manner to the non-EOI case.



Symbol	Min	4541	Max	Description
T_{BB}	100	100	∞	Time between bytes. Talkers must wait long enough to allow the listener to register the end of the transmission of the previous byte, before the next byte can be sent.
T_H	–	N/A	∞	Listener hold-off. *Listener can hold of for any desired period of time, or for no time at all.*
T_{YE}	200	250	∞	EOI indication delay. The sender must wait at least 200 microseconds for the receiver to recognise that a byte is being sent with EOI. If the delay is less than this, the receiver will not acknowledge the EOI.
T_{EI}	60	80	∞	EOI acknowledge hold time. The receiver must pull the DATA line to 0V again for at least 60 microseconds, to allow the sender to detect this EOI acknowledgement pulse.

Symbol	Min	4541	Max	Description
T_{RY}	60	80	100	The talker response limit is a relatively narrow time window during which the talker must commence transmission of the byte. If it is too early, the listener may become desynchronised, because it has not hat time to prepare for reception after sending the EOI acknowledgement. If it is too long, the receiver may conclude the talker has stopped talking.
T_{ST}	20*	35	∞	Data bit setup time. *Referred to as T_S in the C64 Programmer's Reference Guide.*
T_{VT}	20*	35	∞	Data bit valid hold time. *Referred to as T_V in the C64 Programmer's Reference Guide.*
T_F	-	1000	1000	Frame handshake (acknowledge) timeout.
T_{FR}	0/60	0	∞	EOI Frame acknowledge. The listener pulls DATA to 0V after a short period of time to acknowledge the byte sent with EOI.

All time units are in micro seconds, unless otherwise indicated.

* T_{ST} is provided by the peripheral, not the controller. The 4541 is able to respond much faster than a C64 to serial bus events, and thus requires only 4 microsecond to ensure the CLK line has time to rise to 5V. For controllers using software implementations of the protocol, such as the C64, the minimum is 80 microsecond. Therefore peripherals should always use a value of at least 80 microsecond for T_{DA}.

^ The Commodore™ 64 Programmer's Reference Guide lists T_{FR} as having a minimum duration of 60 microsecond. We see no evidence that would suggest that any implementation requires a delay before the DATA line can be pulled low by the listener to acknowledge receipt of a byte.

Receive Byte

Receiving bytes on the IEC bus occurs identically to the sending case. The key difference is that the 4541 is tolerant of a much wider range of timing parameters than software implementations of the bus protocol. The 4541 requires not more than 4

microseconds for any given bus state, which is the time required for the pull-up resistors on the bus to allow lines to float back to 5V. Internally, the 4541 is capable of detecting bit times shorter than 1 microsecond.

OPTIONAL INTEGRATED DATA-LOGGER

The 4541 is available in a variant that contains an embedded serial peripheral bus data-logger. This is designed to aid with debugging protocol errors on this bus. It commences capturing data whenever command $D0 is issued to the 4541, and will capture data for approximately 4 milliseconds.

- **DATALOG0** The low-byte of the optional integrated serial peripheral bus data logger. Writing $00 to this register causes the read pointer to the data log to be reset to the beginning of the capture. Writing $01 to this register, causes the read point of the data logger to be advanced by one time step. The time steps are approximately 1 microsecond, with additional time steps added whenever one of the signals on the serial peripheral bus changes state.
- **DATALOG1** The high-byte of the optional integrated serial peripheral bus data logger. Different fields can be read at this address by writing the following values to the DATALOG0 register:
  - $02 - Read low byte of IEC controller state machine state number.
  - $03 - Read high byte of IEC controller state machine state number.
  - $04 - Read low byte of number of cycles the previous bus state was held for.
  - $05 - Read high byte of number of cycles the previous bus state was held for.

Each time-step records 16 bits of information about the serial peripheral bus:

- **DATALOG0 Bit 0** DATA line input value
- **DATALOG0 Bit 1** CLK line input value
- **DATALOG0 Bit 2** SRQ line input value
- **DATALOG0 Bit 3** DATA line output state
- **DATALOG0 Bit 4** CLK line output state
- **DATALOG0 Bit 5** SRQ line output state
- **DATALOG0 Bit 6** ATN line output state
- **DATALOG0 Bit 7** RESET line output state
- **DATALOG1** Access to the additional data logger fields.

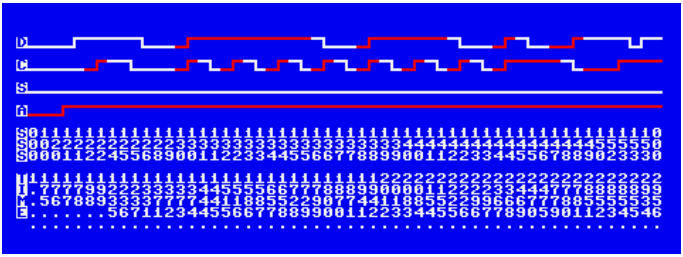
The input values are the voltages that the 4541 reads on the respective pins. Separately, the data-logger records whether the 4541 is pulling each of those lines low. This allows determination as to whether a connected peripheral or the 4541 is pulling a given signal low. This provides much more information than simply probing the lines of the serial peripheral bus, where you cannot readily determine who has pulled a given line low.

The following truth table explains how to interpret these signals:

Input Value	Output Value	Meaning
1	1	Signal is floating at 5V. No device is pulling it low.
0	1	Signal is at 0V. The 4541 is not pulling it low, either a connected peripheral or the CIA is pulling it low.
0	0	Signal is 0V. The 4541 is pulling it low. Other devices might also be pulling it low, but it's not possible to discriminate between these two situations. low.
1	0	Signal is floating at 5V, but the 4541 is pulling it to 0V. Your computer is probably broken, or someone has connected 5V to the signal without a current-limiting resistor in between, in which case, your computer is about to be broken.

Extracting Data from the Data Logger

The following program extracts and displays the contents of the Data Logger as a PETSCII waveform, like the one shown here:



```
10 WK=$50000
20 NL=PEEK($ED)
30 FETCH 3, 4096*2, 0, WK
40 PRINT"♥DUMPING..."
50 POKE$D694,0
60 FORI=0 TO 4095*2 STEP 2
70 POKEWK+I,PEEK($D694):POKE WK+1+I,PEEK($D695)
80 POKE$D694,1
90 NEXT I
100 PRINT"♥PROCESSING..."
110 PRINT"♥RDQQ|CQQ|SQQ|AQQ|SQ|SQ|SQQ|TQ|IQ|MQ|E0000000000000000_";
120 PV%=0:PS%=0:C%=0:DV=1:CV=1:SV=1:AV=1
130 FORD=0 TO 4095*2 STEP 2
140 V%=PEEK(WK+D):S%=PEEK(WK+D+1)
150 IF V%=PV% AND S%=PS% THEN GOTO 460
160 C%=C%+1
170 DD=C%>1 AND DV<>(V%AND$01) : DV=(V% AND $01)
180 CD=C%>1 AND CV<>(V%AND$02) : CV=(V% AND $02)
190 SD=C%>1 AND SV<>(V%AND$04) : SV=(V% AND $04)
200 AD=C%>1 AND AV<>(V%AND$40) : AV=(V% AND $40)
210 IFDV=0THEN BEGIN:IFDD=0 THEN D$="‾":ELSE D$="┌":BEND
220 IFDV>0THEN BEGIN:IFDD=0 THEN D$="_":ELSE D$="└":BEND
230 IFCV=0THEN BEGIN:IFCD=0 THEN C$="‾":ELSE C$="┌":BEND
240 IFCV>0THEN BEGIN:IFCD=0 THEN C$="_":ELSE C$="└":BEND
250 IFSV=0THEN BEGIN:IFSD=0 THEN S$="‾":ELSE S$="┌":BEND
260 IFSV>0THEN BEGIN:IFSD=0 THEN S$="_":ELSE S$="└":BEND
270 IFAV=0THEN BEGIN:IFAD=0 THEN A$="‾":ELSE A$="┌":BEND
280 IFAV>0THEN BEGIN:IFAD=0 THEN A$="_":ELSE A$="└":BEND
290 PRINTCHR$(27);"K";
300 IF(V% AND $08)=0 THEN PRINT"▒"; :ELSE PRINT"▒";
310 PRINT D$
320 PRINT"":PRINTCHR$(27);"K";
330 IF(V% AND $10)=0 THEN PRINT"▒"; :ELSE PRINT"▒";
340 PRINT C$
350 PRINT"":PRINTCHR$(27);"K";
360 IF(V% AND $20)=0 THEN PRINT"▒"; :ELSE PRINT"▒";
370 PRINT S$
380 PRINT"":PRINTCHR$(27);"K";
390 PRINT"▒";
```

```
400 PRINT A$;"■■";
410 GOSUB 550:GOSUB 490
420 PRINT"■■■■■■■■■■■■■■■■";
430 PS%=S%
440 PV%=V%
450 IF C%=77 THEN  0,18:C%=0
460 NEXT D
470 PRINT"■";:FORL=0TONL-3:PRINT"■";:NEXTL
480 END
490 N$=LEFT$(STR$(D/2)+".........",6)
500 PRINT"■";
510 FORI=1TO6
520 PRINTCHR$(27);"K";MID$(N$,I,1)
530 NEXT I
540 RETURN
550 S$=MID$(STR$(S%),2,LEN(STR$(S%))):N$=RIGHT$("000"+S$,3)
560 PRINT"■";
570 FORI=1TO3
580 PRINTCHR$(27);"K";MID$(N$,I,1)
590 NEXT I
600 RETURN
```

CHAPTER 10

Reference Tables

- Units of Storage
- Base Conversion

UNITS OF STORAGE

Unit	Equals	Abbreviation
1 Bit		
1 Nibble	4 Bits	
1 Byte	8 bits	B
1 Kilobyte	1024 B	KB
1 Megabyte	1024KB or 1,048,576 B	MB

BASE CONVERSION

Decimal	Binary	Hexadecimal
0	%0	$0
1	%1	$1
2	%10	$2
3	%11	$3
4	%100	$4
5	%101	$5
6	%110	$6
7	%111	$7
8	%1000	$8
9	%1001	$9
10	%1010	$A
11	%1011	$B
12	%1100	$C
13	%1101	$D
14	%1110	$E
15	%1111	$F
16	%10000	$10
17	%10001	$11
18	%10010	$12
19	%10011	$13
20	%10100	$14
21	%10101	$15
22	%10110	$16
23	%10111	$17
24	%11000	$18
25	%11001	$19
26	%11010	$1A
27	%11011	$1B
28	%11100	$1C
29	%11101	$1D
30	%11110	$1E
31	%11111	$1F

Decimal	Binary	Hexadecimal
32	%100000	$20
33	%100001	$21
34	%100010	$22
35	%100011	$23
36	%100100	$24
37	%100101	$25
38	%100110	$26
39	%100111	$27
40	%101000	$28
41	%101001	$29
42	%101010	$2A
43	%101011	$2B
44	%101100	$2C
45	%101101	$2D
46	%101110	$2E
47	%101111	$2F
48	%110000	$30
49	%110001	$31
50	%110010	$32
51	%110011	$33
52	%110100	$34
53	%110101	$35
54	%110110	$36
55	%110111	$37
56	%111000	$38
57	%111001	$39
58	%111010	$3A
59	%111011	$3B
60	%111100	$3C
61	%111101	$3D
62	%111110	$3E
63	%111111	$3F

Decimal	Binary	Hexadecimal
64	%1000000	$40
65	%1000001	$41
66	%1000010	$42
67	%1000011	$43
68	%1000100	$44
69	%1000101	$45
70	%1000110	$46
71	%1000111	$47
72	%1001000	$48
73	%1001001	$49
74	%1001010	$4A
75	%1001011	$4B
76	%1001100	$4C
77	%1001101	$4D
78	%1001110	$4E
79	%1001111	$4F
80	%1010000	$50
81	%1010001	$51
82	%1010010	$52
83	%1010011	$53
84	%1010100	$54
85	%1010101	$55
86	%1010110	$56
87	%1010111	$57
88	%1011000	$58
89	%1011001	$59
90	%1011010	$5A
91	%1011011	$5B
92	%1011100	$5C
93	%1011101	$5D
94	%1011110	$5E
95	%1011111	$5F

Decimal	Binary	Hexadecimal
96	%1100000	$60
97	%1100001	$61
98	%1100010	$62
99	%1100011	$63
100	%1100100	$64
101	%1100101	$65
102	%1100110	$66
103	%1100111	$67
104	%1101000	$68
105	%1101001	$69
106	%1101010	$6A
107	%1101011	$6B
108	%1101100	$6C
109	%1101101	$6D
110	%1101110	$6E
111	%1101111	$6F
112	%1110000	$70
113	%1110001	$71
114	%1110010	$72
115	%1110011	$73
116	%1110100	$74
117	%1110101	$75
118	%1110110	$76
119	%1110111	$77
120	%1111000	$78
121	%1111001	$79
122	%1111010	$7A
123	%1111011	$7B
124	%1111100	$7C
125	%1111101	$7D
126	%1111110	$7E
127	%1111111	$7F

Decimal	Binary	Hexadecimal
128	%10000000	$80
129	%10000001	$81
130	%10000010	$82
131	%10000011	$83
132	%10000100	$84
133	%10000101	$85
134	%10000110	$86
135	%10000111	$87
136	%10001000	$88
137	%10001001	$89
138	%10001010	$8A
139	%10001011	$8B
140	%10001100	$8C
141	%10001101	$8D
142	%10001110	$8E
143	%10001111	$8F
144	%10010000	$90
145	%10010001	$91
146	%10010010	$92
147	%10010011	$93
148	%10010100	$94
149	%10010101	$95
150	%10010110	$96
151	%10010111	$97
152	%10011000	$98
153	%10011001	$99
154	%10011010	$9A
155	%10011011	$9B
156	%10011100	$9C
157	%10011101	$9D
158	%10011110	$9E
159	%10011111	$9F

Decimal	Binary	Hexadecimal
160	%10100000	$A0
161	%10100001	$A1
162	%10100010	$A2
163	%10100011	$A3
164	%10100100	$A4
165	%10100101	$A5
166	%10100110	$A6
167	%10100111	$A7
168	%10101000	$A8
169	%10101001	$A9
170	%10101010	$AA
171	%10101011	$AB
172	%10101100	$AC
173	%10101101	$AD
174	%10101110	$AE
175	%10101111	$AF
176	%10110000	$B0
177	%10110001	$B1
178	%10110010	$B2
179	%10110011	$B3
180	%10110100	$B4
181	%10110101	$B5
182	%10110110	$B6
183	%10110111	$B7
184	%10111000	$B8
185	%10111001	$B9
186	%10111010	$BA
187	%10111011	$BB
188	%10111100	$BC
189	%10111101	$BD
190	%10111110	$BE
191	%10111111	$BF

Decimal	Binary	Hexadecimal
192	%11000000	$C0
193	%11000001	$C1
194	%11000010	$C2
195	%11000011	$C3
196	%11000100	$C4
197	%11000101	$C5
198	%11000110	$C6
199	%11000111	$C7
200	%11001000	$C8
201	%11001001	$C9
202	%11001010	$CA
203	%11001011	$CB
204	%11001100	$CC
205	%11001101	$CD
206	%11001110	$CE
207	%11001111	$CF
208	%11010000	$D0
209	%11010001	$D1
210	%11010010	$D2
211	%11010011	$D3
212	%11010100	$D4
213	%11010101	$D5
214	%11010110	$D6
215	%11010111	$D7
216	%11011000	$D8
217	%11011001	$D9
218	%11011010	$DA
219	%11011011	$DB
220	%11011100	$DC
221	%11011101	$DD
222	%11011110	$DE
223	%11011111	$DF

Decimal	Binary	Hexadecimal
224	%11100000	$E0
225	%11100001	$E1
226	%11100010	$E2
227	%11100011	$E3
228	%11100100	$E4
229	%11100101	$E5
230	%11100110	$E6
231	%11100111	$E7
232	%11101000	$E8
233	%11101001	$E9
234	%11101010	$EA
235	%11101011	$EB
236	%11101100	$EC
237	%11101101	$ED
238	%11101110	$EE
239	%11101111	$EF
240	%11110000	$F0
241	%11110001	$F1
242	%11110010	$F2
243	%11110011	$F3
244	%11110100	$F4
245	%11110101	$F5
246	%11110110	$F6
247	%11110111	$F7
248	%11111000	$F8
249	%11111001	$F9
250	%11111010	$FA
251	%11111011	$FB
252	%11111100	$FC
253	%11111101	$FD
254	%11111110	$FE
255	%11111111	$FF

CHAPTER 11

Supporters & Donors

- Organisations
- Contributors
- Supporters

The MEGA65 would not have been possible to create without the generous support of many organisations and individuals.

We are still compiling these lists, so apologies if we haven't included you yet. If you know anyone we have left out, please let us know, so that we can recognise the contribution of everyone who has made the MEGA65 possible, and into the great retro-computing project that it has become.

ORGANISATIONS

The MEGA Museum of Electronic Games & Art e.V. Germany

EVERYTHING

Trenz Electronic, Germany

MOTHERBOARD **MANUFACTURING** **SALES**

Hintsteiner, Austria

CASE

GMK, Germany

KEYBOARD

KEVAG Telekom, Germany

WEB HOSTING

CONTRIBUTORS

Andreas Liebeskind
(libi in paradize)
CFO MEGA eV

Thomas Hertzler
(grumpyninja)
USA spokesman

Russell Peake
(rdpeake)
Bug herding

Alexander Nik Petra
(n0d)
Early case design

Ralph Egas
(0-limits)
Business advisor

Lucas Moss
MEGAphone PCB design

Daren Klamer
(Impakt)
Manual proof-reading

Daniël Mantione
(dmantione)
C64 hardware guru

Dr. Canan Hastik
(indica)
Chairwoman MEGA eV

Simon Jameson
(Shallan)
Platform enhancements

Stephan Kleinert
(ubik)
Destroyer of BASIC 10

Wayne Johnson
(sausage)
Manual additions

L. Kleiss
(LAK132)
MegaWAT presentation software

Maurice van Gils
(Maurice)
BASIC 65 example programs

Andrew Owen
(Cheveron)
Keyboard, Sinclair support

Adam Barnes
(amb5l)
HDMI expert and board revision

Wayne Rittimann, Jr.
(johnwayner)
Bug squashing on all levels

SUPPORTERS

3c74ce64
8-Bit Classics
@11110110100
Aaron Smith
Achim Mrotzek
Adolf Nefischer
Adrian Esdaile
Adrien Guichard
Ahmed Kablaoui
Alan Bastian Witkowski
Alan Field
Alastair Paulin-Campbell
Alberto Mercuri
Alexander Haering
Alexander Kaufmann
Alexander Niedermeier
Alexander Soppart
Alfonso Ardire
Amiga On The Lake
André Kudra
André Simeit
André Wösten
Andrea Farolfi
Andrea Minutello
Andreas Behr
Andreas Freier
Andreas Grabski
Andreas Millinger
Andreas Nopper
Andreas Ochs
Andreas Wendel Manufaktur
Andreas Zschunke
Andrew Bingham
Andrew Dixon
Andrew Mondt
Andrzej Hłuchyj
Andrzej Sawiniec
Andrzej Śliwa
Anthony W. Leal
Arkadiusz Bronowicki
Arkadiusz Kwasny
Arnaud Léandre
Arne Drews
Arne Neumann
Arne Richard Tyarks
Axel Klahr
Balaz Ondrej
Barry Thompson
Bartol Filipovic
Benjamin Maas
Bernard Alaiz
Bernhard Zorn
Bieno Marti-Braitmaier
Bigby
Bill LaGrue
Bjoerg Stojalowski
Björn Johannesson
Bjørn Melbøe
Bo Goeran Kvamme
Boerge Noest
Bolko Beutner
Brett Hallen
Brian Gajewski
Brian Green
Brian Juul Nielsen
Brian Reiter
Bryan Pope
Burkhard Franke
Byron Goodman
Cameron Roberton (KONG)
Carl Angervall
Carl Danowski
Carl Stock
Carl Wall
Carlo Pastore
Carlos Silva
Carsten Sørensen
Cenk Miroglu Miroglu
Chang sik Park
Charles A. Hutchins Jr.
Chris Guthrey
Chris Hooper
Chris Stringer
Christian Boettcher
Christian Eick
Christian Gleinser
Christian Gräfe
Christian Heffner
Christian Kersting
Christian Schiller
Christian Streck
Christian Weyer
Christian Wyk
Christoph Haug
Christoph Huck
Christoph Pross
Christopher Christopher
Christopher Kalk
Christopher Kohlert
Christopher Nelson
Christopher Taylor
Christopher Whillock
Claudio Piccinini
Claus Skrepek
Collen Blijenberg
Constantine Lignos
Crnjaninja
Daniel Auger
Daniel Julien
Daniel Lobitz
Daniel O'Connor
Daniel Teicher
Daniel Tootill
Daniel Wedin
Daniele Benetti
Daniele Gaetano Capursi
Dariusz Szczesniak
Darrell Westbury
David Asenjo Raposo
David Dillard
David Gorgon
David Norwood
David Raulo
David Ross
de voughn accooe
Dean Scully
Dennis Jeschke
Dennis Schaffers
Dennis Schierholz

Dennis Schneck
denti
Dick van Ginkel
Diego Barzon
Dierk Schneider
Dietmar Krueger
Dietmar Schinnerl
Dirk Becker
Dirk Wouters
Domingo Fivoli
DonChaos
Donn Lasher
Douglas Johnson
Dr. Leopold Winter
Dusan Sobotka
Earl Woodman
Ed Reilly
Edoardo Auteri
Eduardo Gallardo
Eduardo Luis Arana
Eirik Juliussen Olsen
Emilio Monelli
EP Technical Services
Epic Sound
Erasmus Kuhlmann
ergoGnomik
Eric Hilaire
Eric Hildebrandt
Eric Hill
Eric Jutrzenka
Erwin Reichel
Espen Skog
Evangelos Mpouras
Ewan Curtis
Fabio Zanicotti
Fabrizio Di Dio
Fabrizio Lodi
FARA Gießen GmbH
FeralChild
First Choice Auto's
Florian Rienhardt
Forum64. de
Francesco Baldassarri
Frank Fechner
Frank Glaush
Frank Gulasch
Frank Haaland
Frank Hempel
Frank Koschel
Frank Linhares
Frank Sleeuwaert
Frank Wolf
FranticFreddie
Fredrik Ramsberg
Fridun Nazaradeh
Friedel Kropp
Garrick West
Gary Lake-Schaal
Gary Pearson
Gavin Jones
Geir Sigmund Straume
Gerd Mitlaender
Giampietro Albiero
Giancarlo Valente
Gianluca Girelli
Giovanni Medina
Glen Fraser
Glen R Perye III
Glenn Main
Gordon Rimac
GRANT BYERS
Grant Louth
Gregor Bubek
Gregor Gramlich
Guido Ling
Guido von Gösseln
Guillaume Serge
Gunnar Hemmerling
Günter Hummel
Guy Simmons
Guybrush Threepwood
Hakan Blomqvist
Hans Pronk
Hans-Jörg Nett
Hans-Martin Zedlitz
Harald Dosch
Harri Salokorpi
Harry Culpan
Harry Venema
Heath Gallimore
Heinz Roesner
Heinz Stampfli
Helge Förster
Hendrik Fensch
Henning Harperath
Henri Parfait
Henrik Kühn
Holger Burmester
Holger Sturk
Howard Knibbs
Hubert de Hollain
Huberto Kusters
Hugo Maria Gerardus v.d. Aa
Humberto Castaneda
Ian Cross
IDE64 Staff
Igor Ianov
Igor Kurtes
Immo Beutler
Ingo Katte
Ingo Keck
Insanely Interested Publishing
IT-Dienstleistungen Obsieger
Ivan Elwood
Jaap HUIJSMAN
Jace Courville
Jack Wattenhofer
Jakob Schönpflug
Jakub Tyszko
James Hart
James Marshburn
James McClanahan
James Sutcliffe
Jan Bitruff
Jan Hildebrandt
Jan Iemhoff
Jan Kösters
Jan Peter Borsje
Jan Schulze
Jan Stoltenberg-Lerche
Janne Tompuri
Jannis Schulte
Jari Loukasmäki
Jason Smith
Javier Gonzalez Gonzalez
Jean-Paul Lauque
Jeffrey van der Schilden
Jens Schneider

Jens-Uwe Wessling
Jesse DiSimone
Jett Adams
Johan Arneklev
Johan Berntsson
Johan Svensson
Johannes Fitz
John Cook
John Deane
John Dupuis
John Nagi
John Rorland
John Sargeant
John Traeholt
Jon Sandelin
Jonas Bernemann
Jonathan Prosise
Joost Honig
Jordi Pakey-Rodriguez
Jöre Weber
Jörg Jungermann
Jörg Schaeffer
Jörg Weese
Josef Hesse
Josef Soucek
Josef Stohwasser
Joseph Clifford
Joseph Gerth
Jovan Crnjanin
Juan Pablo Schisano
Juan S. Cardona Iguina
JudgeBeeb
Juliussen Olsen
Juna Luis Fernandez Garcia
Jürgen Endras
Jürgen Herm Stapelberg
Jyrki Laurila
Kai Pernau
Kalle Pöyhönen
Karl Lamford
Karl-Heinz Blum
Karsten Engstler
Karsten Westebbe
katarakt
Keith McComb
Kenneth Dyke
Kenneth Joensson
Kevin Edwards
Kevin Thomasson
Kim Jorgensen
Kim Rene Jensen
Kimmo Hamalainen
Konrad Buryło
Kosmas Einbrodt
Kurt Klemm
Lachlan Glaskin
Large bits collider
Lars Becker
Lars Edelmann
Lars Slivsgaard
Lasse Lambrecht
Lau Olivier
Lee Chatt
Loan Leray
Lorenzo Quadri
Lorenzo Travagli
Lorin Millsap
Lothar James Foss
Lothar Serra Mari
Luca Papinutti
Ludek Smetana
Lukas Burger
Lutz-Peter Buchholz
Luuk Spaetgens
Mad Web Skills
MaDCz
Magnus Wiklander
Maik Diekmann
Malte Mundt
Manfred Wittemann
Manuel Beckmann
Manzano Mérida
Marc "3D-vice" Schmitt
Marc Bartel
Marc Jensen
Marc Schmidt
Marc Theunissen
Marc Tutor
Marc Wink
Marcel Buchtmann
Marcel Kante
Marco Beckers
Marco Cappellari
Marco Rivela
Marco van de Water
Marcus Gerards
Marcus Herbert
Marcus Linkert
Marek Pernicky
Mario Esposito
Mario Fetka
Mario Teschke
Mariusz Tymków
Mark Adams
Mark Anderson
Mark Green
Mark Hucker
Mark Leitiger
Mark Spezzano
Mark Watkin
Marko Rizvic
Markus Bieler
Markus Bonet
Markus Dauberschmidt
Markus Fehr
Markus Fuchs
Markus Guenther-Hirn
Markus Liukka
Markus Merz
Markus Roesgen
Markus Uttenweiler
Martin Bauhuber
Martin Benke
Martin Gendera
Martin Groß
Martin Gutenbrunner
Martin Johansen
Martin Marbach
Martin Sonnleitner
Martin Steffen
Marvin Hardy
Massimo Villani
Mathias Dellacherie
Mathieu Chouinard
Matthew Adams
Matthew Browne
Matthew Carnevale
Matthew Palmer

Matthew Santos
Matthias Barthel
Matthias Dolenc
Matthias Fischer
Matthias Frey
Matthias Grandis
Matthias Guth
Matthias Lampe
Matthias Meier
Matthias Mueller
Matthias Nofer
Matthias Schonder
Maurice Al-Khaliedy
Max Ihlenfeldt
Meeso Kim
Michael Dailly
Michael Dötsch
Michael Dreßel
Michael Fichtner
Michael Fong
Michael Geoffrey Stone
Michael Gertner
Michael Grün
Michael Habel
Michael Härtig
Michael Haynes
Michael J Burkett
Michael Jensen
Michael Jurisch
Michael Kappelgaard
Michael Kleinschmidt
Michael Lorenz
Michael Mayerhofer
Michael Nurney
Michael Rasmussen
Michael Richmond
Michael Sachse
Michael Sarbak
Michael Schneider
Michael Scholz
Michael Timm
Michael Traynor
Michael Whipp
Michal Ursiny
Michele Chiti
Michele Perini
Michele Porcu
Miguel Angel Rodriguez Jodar
Mikael Lund
Mike Betz
Mike Kastrantas
Mike Pikowski
Mikko Hämäläinen
Mikko Suontausta
Mirko Roller
Miroslav Karkus
Morgan Antonsson
Moritz
Morten Nielsen
MUBIQUO APPS,SL
Myles Cameron-Smith
Neil Moore
Nelson
neoman
Nicholas Melnick
Nikolaj Brinch Jørgensen
Nils Andreas
Nils Eilers
Nils Hammerich
Nils77
Norah Smith
Norman King
Normen Zoch
Olaf Grunert
Ole Eitels
Oliver Boerner
Oliver Brüggmann
Oliver Graf
Oliver Smith
Olivier Bori
ONEPSI LLC
oRdYNe
Osaühing Trioflex
OSHA-PROS USA
Padawer
Patrick Becher
Patrick Bürckstümmer
Patrick de Zoete
Patrick Toal
Patrick Vogt
Paul Alexander Warren
Paul Gerhardt (KONG)
Paul Jackson
Paul Johnson
Paul Kuhnast (mindrail)
Paul Massay
Paul Westlake
Paul Woegerer
Pauline Brasch
Paulo Apolonia
Pete Collin
Pete of Retrohax.net
Peter Eliades
Peter Gries
Peter Habura
Peter Herklotz
Peter Huyoff
Peter Knörzer
Peter Leswell
Peter Weile
Petri Alvinen
Philip Marien
Philip Timmermann
Philipp Rudin
Pierre Kressmann
Pieter Labie
Piotr Kmiecik
Power-on.at
Przemysław Safonow
Que Labs
R Welbourn
R-Flux
Rafał Michno
Rainer Kappler
Rainer Kopp
Rainer Weninger
Ralf Griewel
Ralf Pöscha
Ralf Reinhardt
Ralf Schenden
Ralf Smolarek
Ralf Zenker
Ralph Bauer
Ralph Wernecke
Rédl Károly
Reiner Lanowski
Remi Veilleux
Riccardo Bianchi

Richard Englert
Richard Good
Richard Menedetter
Richard Sopuch
Rick Reynolds
Rico Gruninger
Rob Dean
Robert Bernardo
Robert Eaglestone
Robert Grasböck
Robert Miles
Robert Schwan
Robert Shively
Robert Tangmar
Robert Trangmar
Rodney Xerri
Roger Olsen
Roger Pugh
Roland Attila Kett
Roland Evers
Roland Schatz
Rolf Hass
Ronald Cooper
Ronald Hunn
Ronny Hamida
Ronny Preiß
Roy van Zundert
Rüdiger Wohlfromm
Ruediger Schlenter
Rutger WIllemsen
Sampo Peltonen
Sarmad Gilani
SAS74
Sascha Hesse
Scott Halman
Scott Hollier
Scott Robison
Sebastian Baranski
Sebastian Bölling
Sebastian Felzmann
Sebastian Lipp
Sebastian Rakel
Şemseddin Moldibi
Śeth Morabito
Shawn McKee
Siegfried Hartmann
Zytex Online Store
Sigurbjorn Larusson
Sigurdur Finnsson
Simon Lawrence
Simon Wolf
spreen.digital
Stefan Haberl
Stefan Kramperth
Stefan Richter
Stefan Schultze
Stefan Sonnek
Stefan Theil
Stefan Vrampe
Stefano Canali
Stefano Mozzi
Steffen Reiersen
Stephan Bielmann
Stephen Jones
Stephen Kew
Steve Gray
Steve Kurlin
Steve Lemieux
Steven Combs
Stewart Dunn
Stuart Marsh
Sven Neumann
Sven Stache
Sven Sternberger
Sven Wiegand
Szabolcs Bence
Tantrumedia Limited
Techvana Operations Ltd.
Teddy Turmeaux
Teemu Korvenpää
The Games Foundation
Thierry Supplisson
Thieu-Duy Thai
Thomas Bierschenk
Thomas Edmister
Thomas Frauenknecht
Thomas Gitzen
Thomas Gruber
Thomas Haidler
Thomas Jager
Thomas Karlsen
Thomas Laskowski
Thomas Marschall
Thomas Niemann
Thomas Scheelen
Thomas Schilling
Thomas Tahsin-Bey
Thomas Walter
Thomas Wirtzmann
Thorsten Knoll
Thorsten Nolte
Tim Krome
Tim Waite
Timo Weirich
Timothy Blanks
Timothy Henson
Timothy Prater
Tobias Butter
Tobias Heim
Tobias Köck
Tobias Lüthi
Tommi Vasarainen
Toni Ammer
Tore Olsen
Torleif Strand
Torsten Schröder
Tuan Nguyen
Uffe Jakobsen
Ulrich Hintermeier
Ulrich Nieland
Ulrik Kruse
Urban Lindeskog
Ursula Förstle
Uwe Anfang
Uwe Boschanski
Vedran Vrbanc
Verm Project
Wayne Rittimann
Wayne Sander
Wayne Steele
Who Knows
Winfried Falkenhahn
Wolfgang Becker
Wolfgang Stabla
Worblehat
www.patop69.net
Yan B
Zoltan Markus
Zsolt Zsila

Bibliography

[1] L. Soares and M. Stumm, "Flexsc: Flexible system call scheduling with exception-less system calls." in *Osdi*, vol. 10, 2010, pp. 1–8.

[2] N. Montfort, P. Baudoin, J. Bell, I. Bogost, J. Douglass, M. C. Marino, M. Mateas, C. Reas, M. Sample, and N. Vawter, *10 PRINT CHR $(205.5+ RND (1));: GOTO 10*. MIT Press, 2012.

[3] Actraiser, "Vic-ii for beginners: Screen modes, cheaper by the dozen," 2013. [Online]. Available: http://dustlayer.com/vic-ii/2013/4/26/vic-ii-for-beginners-screen-modes-cheaper-by-the-dozen

INDEX

53301, 52
53302, 52
53303, 52
53304, 52
53305, 52
53306, 52
53307, 52
53308, 52
53309, 52
53310, 52
53311, 53
53312, 53
53313, 53
53314, 53
53315, 53
53316, 53
53317, 53
53318, 53
53319, 53
53320, 54
53321, 54
53322, 54
53323, 55
53324, 55
53325, 55
53326, 55
53327, 55
53328, 55
53329, 55
53330, 55
53331, 55
53332, 55
53333, 55
53334, 55
53335, 55
53336, 55
53337, 55
53338, 55
53339, 55
53340, 55
53341, 55
53342, 55
53343, 55
53344, 55
53345, 55
53346, 55
53347, 55
53348, 55
53349, 55
53352, 55
53353, 55
53354, 55
53355, 55
53356, 55
53357, 55
53358, 55
53359, 55
53360, 55
53361, 55
53362, 55
53363, 55
53364, 55
53365, 56
53366, 56
53367, 56
53368, 56
53369, 56
53370, 56
53371, 56
53372, 56
53376, 128
53377, 128
53378, 128
53379, 128
53380, 128
53381, 128
53382, 128
53383, 128
53384, 128
53385, 128
53386, 128
54272, 63
54273, 63
54274, 63