MEGA65

USER'S GUIDE

MUSEUM OF ELECTRONIC GAMES & ART

REGULATORY INFORMATION

The MEGA65 home computer and portable computer have not been subject to FCC, EC or other regulatory approvals as of the time of writing.

MEGA65 REFERENCE GUIDE

Published by
the MEGA Museum of Electronic Games and Art, Germany.
and
Flinders University, Australia.

WORK IN PROGRESS

November 13, 2019

Contents

PART I

APPENDICES

APPENDICES

APPENDIX A

45GS02 Microprocessor

- Introduction
- Differences to the 6502
- C64 CPU Memory Mapped Registers
- New CPU Memory Mapped Registers
- MEGA65 CPU Math Unit Registers
- MEGA65 Hypervisor Mode

INTRODUCTION

The 45GS02 is an enhanced version of the processor portion of the CSG4510 and of the F018 "DMAgic" DMA controller used in the Commodore 65 computer prototypes. The 4510 is, in turn, an enhanced version of the 65CE02. The reader is referred to the considerable documentation available for the 6502 and 65CE02 processors for the backwards-compatibile operation of the 45GS02.

This chapter will focus on the differences between the 45GS02 and the earlier 6502-class processors, and the documentation of the many built-in memory-mapped IO registers of the 45GS02.

DIFFERENCES TO THE 6502

The 45GGS02 has a number of key differences to earlier 6502-class processors:

Supervisor/Hypervisor Privileged Mode

Unlike the earlier 6502 variants, the 45GS02 has a privileged mode of operation. This mode is intended for use by an operating system or type-1 hypervisor. The ambiguity between operating system and hypervisor on the MEGA65 stems from the fact that the operating system of the MEGA65 is effectively little more than a loader and task-switcher for C64 and C65 environments, i.e., effectively operating as a hypervisor, but provides only limited virtualisation of the hardware.

The key differences between normal and supervisor mode on the MEGA65, are that in supervisor mode:

- A special 16KiB memory area is mapped to $8000 - $BFFF, which is used to contain both the program and data of the hypervisor / supervisor program. This is normally the Hyppo program. This memory is not mappable by any means when the processor is in the normal mode (the chip-select line to it is inhibited), protecting it from accidental or malicious access.
- The 64 SYSCALL trap registers in the MEGA65 IO-mode at $D640 - $D67F are replaced by the virtualisation control registers. These registers allow complete control over the system, and it is their access that truly defines the privilege of the supervisor mode.
- The processor always operates at full speed (40MHz) and in the 4510 processor personality.

The Hypervisor Mode is described in more detail later in this appendix.

6502 Illegal Opcodes

The 65C02, 65CE02 and CSG4510 processors extended the original 6502 processor by using previously unallocated opcodes of the 6502 to provide additional instructions. All software that followed the official documentation of the 6502 processor will therefore work on these newer processors, possibly with different instruction timing. However, the common practice on the C64 and other home computers of using undefined opcodes (often called "illegal opcodes", although there is no law against using them), means that many existing programs will not work on these newer processors.

To alieviate this problem the 45GS02 has the ability to switch processor personalities between the 4510 and 6502. The effect is that in 6502 mode, none of the new opcodes of the 65C02, 65CE02, 4510 or 45GS02 are available, and are replaced with the original, often strange, behaviour of the undefined opcodes of the 6502.

WARNING: This feature is incomplete and untested. Most undocumented 6502 opcodes do not operate correctly when the 6502 personality is enabled.

Read-Modify-Write Instruction Bug Compatibility

The 65CE02 processor optimised a group of instructions called the Read-Modify-Write (RMW) instructions. For such instructions, such as INC, that increments the contents of a memory location, the 6502 would read the original value and then write it back unchanged, before writing it back with the new increased value. For most purposes, this did not cause any problems. However, it turned out to be a fast way to acknowledge VIC-II interrupts, because writing the original value back (which the instruction doesn't need to do) acknowledges the interrupt. This method is faster and uses fewer bytes than any alternative, and so became widely used in C64 software.

The problem came with the C65 with its 65CE02 derived CSG4510 that didn't do this extra write during the RMW instructions. This made the RMW instructions one cycle faster, which made software run slightly faster. Unfortunately, it also meant that a lot of existing C64 software simply won't run on a C65, unless the interrupt acknowledgement code in each program is patched to work around this problem. This is the single most common reason why many C64 games and other software titles won't run on a C65.

Because this problem is so common, the MEGA65's 45GS02 includes bug compatibility with this commonly used feature of the original 6502. It does this by checking if the target of an RMW instruction is $D019, i.e., the interrupt status register of the VIC-II. If it is, then the 45GS02 performs the dummy write, allowing many C64 software titles to run unmodified on the MEGA65, that do not run on a C65 prototype. By only performing the dummy write if the address is $D019, the MEGA65 maintains C64 compatibility, without sacrificing the speed improvement for all other uses of these instructions.

Variable CPU Speed

The 45GSG02 is able to run at 1MHz, 2MHz, 3.5MHz and 40MHz, to support running software designed for the C64, C128 in C64 mode, C65 and MEGA65.

Slow (1MHz – 3.5MHz) Operation

In these modes, the 45GS02 processor slows down, so that the same number of instructions per video frame are executed as on a PAL or NTSC C64, C128 in C64 mode or C65 prototype. This is to allow existing software to run on the MEGA65 at the correct speed, and with minimal display problems. The VIC-IV video controller provides cycle indication pulses to the 45GS02 that are used to keep time.

In these modes, opcodes take the same number of cycles as an 6502. However memory accesses within an instruction are not guaranteed to occur in the same cycle as on a 1MHz 6502. Normally the effect is that instructions complete faster, and the processor idles until the correct number of cycles have passed. This means that timing may be incorrect by upto 7 micro-seconds. This is not normally a problem, and even many C64 fast loaders will function correctly. For example, the GEOS(tm) Graphical Operating System for the C64 can be booted and used from a 1541 connected to the MEGA65's serial port.

However, some advanced VIC-II graphics tricks, such as Variable Screen Position (VSP) are highly unlikely to work correctly, due to the uncertainty in timing of the memory write cycles of instructions. However, in most cases such problems can be easily solved by using the advanced features of the MEGA65's VIC-IV video controller. For example, VSP is unnecessary on the MEGA65, because you can set the screen RAM address to any location in memory.

Full Speed (40MHz) Instruction Timing

When the MEGA65's processor is operating at full speed (currently 40MHz), the instruction timing no longer exactly mirrors the 6502: Instructions that can be executed in fewer cycles will do so. For example, branches are typically require fewer instructions on the 45GS02. There are also some instructions that require more cycles on the 45GS02, in particular the LDA, LDX, LDY and LDZ instructions. Those instructions typically require one additional cycle. However as the processor is running at 40MHz, these instructions still execute much more quickly than on even a C65 or C64 with an accelerator.

CPU Speed Fine-Tuning

It is also possible to more smoothly vary the CPU speed using the **SPEEDBIAS** register located at $F7FA (55290), when MEGA65 IO mode is enabled. The default value is $80 (128), which means no bias on the CPU speed. Higher values increase the CPU speed, with $FF meaning $2\times$ the expected speed. Lower values slow the processor down, with

$00 bring the CPU to a complete stand-still. Thus the speed can be varied between $0\times$ and $2\times$ the inteded value.

This register is provided to allow tweaking the processor speed in games.

Note that this register has no effect when the processor is running at full-speed, because it only affects the way in which VIC-IV video cycle indication pulses are processed by the CPU.

Direct Memory Access (DMA)

Direct Memory Access (DMA) is a method for quickly filling, copying or swapping memory regions. The MEGA65 implements an improved version of the F018 "DMAgic" DMA controller of the C65 prototypes. Unlike on the C65 prototypes, the DMA controller is part of the CPU on the MEGA65.

Detailed information on how to use the DMA controller and these advanced features can be found in Appendix **??**

Accessing memory between the 64KB and 1MB points

The C65 included four ways to access memory beyond the 64KB point: three methods that are limited, specialised or both, and two general-purpose methods. We will first consider the limited methods, before documenting the general-purpose methods.

C64-Style Memory Banking

The first method, is to use the C64-style $00/$01 ROM/RAM banking. This method is very limited, however, as it allows only the banking in and out of the two 8KB regions that correspond to the C64 BASIC and KERNAL ROMs. These are located at $2A000 and $2E000 in the 20-bit C65 address space, i.e., $002A000 and $002E000 in the 28-bit address space of the MEGA65. It can also provide access to the C64 character ROM data at $D000, which is located at $2D000 in the C65 memory map, and thus $002D0000 in the MEGA65 address space. In addition to being limited to which regions this method can access, it also only provides read-only access to these memory regions, i.e., it cannot be used to modify these memory regions.

VIC-III "ROM" Banking

Similar to the C64-style memory banking, the C65 included the facility to bank several other regions of the C65's 128KB ROM. These are banked in and out using various bits of the VIC-III's $D030 register:

$D030 Bit	Signal Name	20-bit Address	16-bit Address	Read-Write Access?
0	CRAM2K	$1F800 – $1FFFF, $FF80000 – $FF807FF	$D800 – $DFFF	Y
3	ROM8	$38000 – $39FFF	$8000 – $9FFF	N
4	ROMA	$3A000 – $3BFFF	$A000 – $BFFF	N
5	ROMC	$2C000 – $2CFFF	$C000 – $CFFF	N
6	CROM9	$29000 – $29FFF	$D000 – $DFFF	N
7	ROME	$3E000 – $3FFFF	$E000 – $FFFF	N

The CRAM2K signal causes the normal 1KB of colour RAM, which is located at $1F800 – $1FBFF and is visible at $D800 – $DBFF, to instead be visible from $D800 – $DFFF. That is, the entire range $1F800 – $1FFFF is visible, and can be both read from and written to. Unlike on the C64, the colour RAM on the MEGA65 is always visible as 8-bit bytes. Also, on the MEGA65, the colour RAM is 32KB in size, and exists at $FF80000 – $FF87FFF. The visibility of the colour RAM at $1F800 – $1FFFF is achieved by mirroring writes to both regions when accessing the colour RAM via this mechanism.

Note that these VIC-III memory banking signals take precedence over the C64-style memory banking.

VIC-III Display Address Translator

The third specialised manner to access to memory abover the 64KB point is to use the VIC-III's Display Address Translator. Use of this mechanism is documented in Appendix B.

The MAP instruction

The first general-purpose means of access to memory is the MAP instruction of the 4510 processor. The MEGA65's 45GS02 processor also supports this mechanism. This instruction divides the 64KB address of the 6502 into eight blocks of 8KB each. For each of these blocks, the block may either be accessed normally, i.e., accessing an 8KB region of the first 64KB of RAM of the system. Alternatively, each block may instead be re-mapped (hence the name of the MAP instruction) to somewhere else in the address space, by adding an offset to the address. Mapped addresses in the first 32KB use one offset, the lower offset, and the second 32KB uses another, the upper offset. Re-mapping of memory using the MAP instruction takes precedence over the C64-style memory banking, but not the C65's ROM banking mechanism.

The offsets must be a multiple of 256 bytes, and thus consist of 12 bits in order to allow an arbitrary offset in the 1MB address space of the C65. As each 8KB block in a 32KB half of memory can be either mapped or not, this requires one bit per 8KB block. Thus the processor requires 16 bits of information for each half of memory, for a total of 32 bits of information. This is achieved by setting the A and X registers for the lower half of memory and the Y and Z registers for the upper half of memory, before executing the MAP instruction. The MAP instruction copies the contents of these registers into the processors internal registers that hold the mapping information. Note that there is no way to use the MAP instruction to determine the current memory mapping configuration, which somewhat limits its effectiveness.

See under "Using the MAP instruction to access >1MB" for further explanation and an example.

Direct Memory Access (DMA) Controller

The C65's F018/F018A DMA controller allows for rapid filling, copying and swapping of the contents of memory anywhere in the 1MB address space. Detailed information about the F018 DMA controller, and the MEGA65's enhancements to this, refer to Appendix **??**

Flat Memory Access

Accessing memory beyond the 1MB point

The MEGA65 can support up to 256MiB of memory. This is more than the 1MiB address space of the CSG4510 on which it is based. There are several ways of performing this.

Using the MAP instruction to access >1MB

The full address space is available to the MAP instruction for legacy C65-style memory mapping, although some care is required, as the MAP instruction must be called up to three times. The reason for this is that the MAP instruction must be called to first select which mega-byte of memory will be used for the lower and upper map regions, before it is again called in the normal way to set the memory mapping. Because between these two calls the memory mapping offset will be a mix of the old and new addresses, all mapping should be first disabled via the MAP instruction. This means that the code to re-map memory should live in the bottom 64KB of RAM or in one of the ROM-bankable regions, so that it can remain visible during the mapping process.

Failure to handle this situation properly will result in the processor executing instructions from somewhere unexpected half-way through the process, because the routine it is executing to perform the mapping will suddenly no longer be mapped.

Because of the relative complexity of this process, and the other problems with the MAP instruction as a means of memory access, we recommend that for accessing data outside of the current memory map that you use either DMA or the flat-memory address features of the 45GS02 that are described below. Indeed, access to the full address space via the MAP instruction is only provided for completeness.

To give a very simple example of how the MAP instruction can be used to map an area of memory from the expanded address space, the following program maps the ethernet frame buffer from its natural location at $FFDE8000 to appear at $6800. To keep the example as simple as possible, we assume that the code is running from in the bottom 64KB of RAM, and not in the region between $6000 – $8000.

As the MAP instruction normally is only aware of the C65-style 20-bit addresses, the MEGA65 extension to the instruction must be used to set the upper 8 bits of the 28-bit MEGA65 addresses, i.e., which mega-byte of address space should be used for the address translation. This is done by setting the X register to $0F when setting the mega-byte number for the lower-32KB of the C64-style 64KB address space. This does not create any incompatibility with any sensible use of the MAP instruction on a C65, because this value indicates that none of the four 8KB memory blocks will be re-mapped, but at the same time specifies that the upper 4 bits of the address offset for re-mapped block is the non-zero value of $F. The mega-byte number is then specified by setting the A register.

The same approach applie to the upper 32KB, but using the Z and Y registers instead of the X and A registers. However, in this case, we do not need to re-map the upper 32KB of memory in this example, we will leave the Z and Y registers set to zero. We must however set X and A to set the mega-byte number for the lower-32KB to $FF. Therefore A must have the value $FF. To set the lower 20-bits of the address offset we use the MAP instruction a second time, this time using it in the normal C65 manner. As we want to remap $6800 to $FFDE800, and have already dealt with the $FFxxxxx offset via the mega-byte number, we need only to apply the offset to make $6800 point to $DE800. $DE800 minus $6800 = $D8000. As the MAP instruction operates with a mapping granularity of 256 bytes = $100, we can drop the last two digits from $D8000 to obtain the MAP offset of $D80. The lower 8-bits, $80, must be loaded into the A register. The upper 4-bits, $D, must be loaded into the low-nybl of the X register. As we wish to apply the mapping to only the fourth of the 8KB blocks that make up the lower 32KB half of the C64 memory map, we must set the 4th bit of the upper nybl. That is, the upper nybl must be set to %1000, i.e., $8. Therefore the X register must be loaded with $8D. Thus we yield the complete example program:

```
; Map ethernet registers at $6000 - $7FFF
; Ethernet controller really lives $FFDE000 - $FFDEFFF, so select $FF megabyte section for MA
lda #$ff
ldx #$0f
ldy #$00
ldz #$00
map

; now enable mapping of $DE000-$DFFFF at $6000
; MAPs are offset based, so we need to subtract $6000 from the target address
; $DE000 - $6000 = $D8000
lda #$80
ldx #$8d
ldy #$00
ldz #$00
map
eom

; Ethernet buffer now visible at $6800 - $6FFF
```

Note that the EOM (End Of Mapping) instruction (which is the same as NOP on a 6502, i.e., opcode $EA) was only supplied after the last MAP instruction, to make sure that no interrupts could occur while the memory map contained mixed values with the mega-byte number set, but the lower-bits of the mapping address had not been updated.

No example in BASIC for the MAP instruction is possible, because the MAP is an machine code instruction of the 4510 / 45GS02 processors.

Flat-Memory Access

The 45GS02 makes it easy to read or write a byte from anywhere in memory by allowing the Zero-Page Indirect addressing mode to use a 32-bit pointer instead of the normal 16-bit pointer. This is accomplished by using the Z-indexed Zero-Page Indirect Addressing Mode for the access, and having the instruction directly preceeded by a NOP instruction (opcode $EA). For example:

```
NOP
LDA ($45),Z
```

Would read the four bytes of Zero-Page memory at $45 – $48 to form a 32-bit memory address, and add the value of the Z register to this to form the actual address that will be read from. The byte order in the address is the same as the 6502, i.e., the right-most (least significant) byte of the address will be read from the first address ($45 in this case), and so on, until the left-most (most significant) byte will be read from $48. For example, to read from memory location $12345678, the contents of memory beginning at $45 should be 78 56 43 12.

This method iss much more efficient and also simpler than either using the MAP instruction or the DMA controller for single memory accesses, and is what we generally recommend. The DMA controller can be used for moving/filler larger regions of memory. We recommend the MAP instruction only be used for banking code, or in rare situations where extensive access to a small region of memory is required, and the extra cycles of reading the 32-bit addresses is problematic.

Virtual 32-bit Register

The 45GS02 allows the use of its four general purpose registers, A, X ,Y and Z as a single virtual 32-bit register. This can greatly simplify and speed up many common operations, and help avoid many common programming errors. For example, adding two 16-bit or 32-bit values can now be easily accomplished with something like:

```
; Clear carry before performing addition, as normal
CLC
; Prefix an instruction with two NEG instructions to select virtual 32-bit register mode
NEG
NEG
LDA $1234  ; Load the contents of $1234-$1237 into A,X,Y and Z respectively
; And again, for the addition
NEG
NEG
ADC $1238  ; Add the contents of $1238-$123B
; The result of the addition is now in A, X, Y and Z.
; And can be written out in whole or part

; To write it all out, again, we need the NEG + NEG prefix
NEG
NEG
STA $123C ; Write the whole out to $123C-$123F

; Or to write out the bottom bytes, we can just write the contents of A and X as normal
```

```
    STA $1240
    STX $1241
```

This approach works with the LDA, STA, ADC, SBC, CMP, EOR, AND, ORA, ASL, LSR, ROL, ROR, INC and DEC instructions. It also works with any addressing mode. Indexed addressing modes, where X, Y or Z are added to the address should be used with care, because these registers may in fact be holding part of a 32-bit value. The special case is the Zero-Page Indirect Z-Indexed addressing mode: In this case the Z register is NOT added to the target address, as would normally be the case. This is to allow the virtual 32-bit register to be able to be used with flat-memory access with the combined prefix of NEG NEG NOP before the instruction to allow accessing a 32-bit value anywhere in memory in a single instruction.

Note that the virtual 32-bit register cannot be used in immediate mode, i.e., to load a constant into the four general purpose registers. This is to avoid problems with variable length instructions. Also, it would not save any bytes compared to LDA #$nn ... LDZ #$nn, and would be no faster.

C64 CPU MEMORY MAPPED REGISTERS

HEX	**DEC**	**Signal**	**Description**
0000	0	PORTDDR	6510/45GS10 CPU port DDR
0001	1	PORT	6510/45GS10 CPU port data

NEW CPU MEMORY MAPPED REGISTERS

HEX	**DEC**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**
D640	54848	HTRAP00							
D641	54849	HTRAP01							
D642	54850	HTRAP02							
D643	54851	HTRAP03							
D644	54852	HTRAP04							
D645	54853	HTRAP05							
D646	54854	HTRAP06							
D647	54855	HTRAP07							
D648	54856	HTRAP08							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D649	54857	HTRAP09							
D64A	54858	HTRAP0A							
D64B	54859	HTRAP0B							
D64C	54860	HTRAP0C							
D64D	54861	HTRAP0D							
D64E	54862	HTRAP0E							
D64F	54863	HTRAP0F							
D650	54864	HTRAP10							
D651	54865	HTRAP11							
D652	54866	HTRAP12							
D653	54867	HTRAP13							
D654	54868	HTRAP14							
D655	54869	HTRAP15							
D656	54870	HTRAP16							
D657	54871	HTRAP17							
D658	54872	HTRAP18							
D659	54873	HTRAP19							
D65A	54874	HTRAP1A							
D65B	54875	HTRAP1B							
D65C	54876	HTRAP1C							
D65D	54877	HTRAP1D							
D65E	54878	HTRAP1E							
D65F	54879	HTRAP1F							
D660	54880	HTRAP20							
D661	54881	HTRAP21							
D662	54882	HTRAP22							
D663	54883	HTRAP23							
D664	54884	HTRAP24							
D665	54885	HTRAP25							
D666	54886	HTRAP26							
D667	54887	HTRAP27							
D668	54888	HTRAP28							
D669	54889	HTRAP29							
D66A	54890	HTRAP2A							
D66B	54891	HTRAP2B							
D66C	54892	HTRAP2C							
D66D	54893	HTRAP2D							
D66E	54894	HTRAP2E							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D66F	54895	HTRAP2F							
D670	54896	HTRAP30							
D671	54897	HTRAP31							
D672	54898	HTRAP32							
D673	54899	HTRAP33							
D674	54900	HTRAP34							
D675	54901	HTRAP35							
D676	54902	HTRAP36							
D677	54903	HTRAP37							
D678	54904	HTRAP38							
D679	54905	HTRAP39							
D67A	54906	HTRAP3A							
D67B	54907	HTRAP3B							
D67C	54908	HTRAP3C							
D67D	54909	HTRAP3D							
D67E	54910	HTRAP3E							
D67F	54911	HTRAP3F							
D7FA	55290	SPEEDBIAS							
D7FB	55291	-							BRCOST
D7FD	55293	NOEXROM	NOGAME	-					

- **BRCOST** 1=charge extra cycle(s) for branches taken
- **HTRAP00** Writing triggers hypervisor trap $00
- **HTRAP01** Writing triggers hypervisor trap $01
- **HTRAP02** Writing triggers hypervisor trap $02
- **HTRAP03** Writing triggers hypervisor trap $03
- **HTRAP04** Writing triggers hypervisor trap $04
- **HTRAP05** Writing triggers hypervisor trap $05
- **HTRAP06** Writing triggers hypervisor trap $06
- **HTRAP07** Writing triggers hypervisor trap $07
- **HTRAP08** Writing triggers hypervisor trap $08
- **HTRAP09** Writing triggers hypervisor trap $09
- **HTRAP0A** Writing triggers hypervisor trap $0A

- **HTRAP0B** Writing triggers hypervisor trap $0B
- **HTRAP0C** Writing triggers hypervisor trap $0C
- **HTRAP0D** Writing triggers hypervisor trap $0D
- **HTRAP0E** Writing triggers hypervisor trap $0E
- **HTRAP0F** Writing triggers hypervisor trap $0F
- **HTRAP10** Writing triggers hypervisor trap $10
- **HTRAP11** Writing triggers hypervisor trap $11
- **HTRAP12** Writing triggers hypervisor trap $12
- **HTRAP13** Writing triggers hypervisor trap $13
- **HTRAP14** Writing triggers hypervisor trap $14
- **HTRAP15** Writing triggers hypervisor trap $15
- **HTRAP16** Writing triggers hypervisor trap $16
- **HTRAP17** Writing triggers hypervisor trap $17
- **HTRAP18** Writing triggers hypervisor trap $18
- **HTRAP19** Writing triggers hypervisor trap $19
- **HTRAP1A** Writing triggers hypervisor trap $1A
- **HTRAP1B** Writing triggers hypervisor trap $1B
- **HTRAP1C** Writing triggers hypervisor trap $1C
- **HTRAP1D** Writing triggers hypervisor trap $1D
- **HTRAP1E** Writing triggers hypervisor trap $1E
- **HTRAP1F** Writing triggers hypervisor trap $1F
- **HTRAP20** Writing triggers hypervisor trap $20
- **HTRAP21** Writing triggers hypervisor trap $21
- **HTRAP22** Writing triggers hypervisor trap $22
- **HTRAP23** Writing triggers hypervisor trap $23
- **HTRAP24** Writing triggers hypervisor trap $24
- **HTRAP25** Writing triggers hypervisor trap $25
- **HTRAP26** Writing triggers hypervisor trap $26

- **HTRAP27** Writing triggers hypervisor trap $27
- **HTRAP28** Writing triggers hypervisor trap $28
- **HTRAP29** Writing triggers hypervisor trap $29
- **HTRAP2A** Writing triggers hypervisor trap $2A
- **HTRAP2B** Writing triggers hypervisor trap $2B
- **HTRAP2C** Writing triggers hypervisor trap $2C
- **HTRAP2D** Writing triggers hypervisor trap $2D
- **HTRAP2E** Writing triggers hypervisor trap $2E
- **HTRAP2F** Writing triggers hypervisor trap $2F
- **HTRAP30** Writing triggers hypervisor trap $30
- **HTRAP31** Writing triggers hypervisor trap $31
- **HTRAP32** Writing triggers hypervisor trap $32
- **HTRAP33** Writing triggers hypervisor trap $33
- **HTRAP34** Writing triggers hypervisor trap $34
- **HTRAP35** Writing triggers hypervisor trap $35
- **HTRAP36** Writing triggers hypervisor trap $36
- **HTRAP37** Writing triggers hypervisor trap $37
- **HTRAP38** Writing triggers hypervisor trap $38
- **HTRAP39** Writing triggers hypervisor trap $39
- **HTRAP3A** Writing triggers hypervisor trap $3A
- **HTRAP3B** Writing triggers hypervisor trap $3B
- **HTRAP3C** Writing triggers hypervisor trap $3C
- **HTRAP3D** Writing triggers hypervisor trap $3D
- **HTRAP3E** Writing triggers hypervisor trap $3E
- **HTRAP3F** Writing triggers hypervisor trap $3F
- **NOEXROM** Override for /EXROM : Must be 0 to enable /EXROM signal
- **NOGAME** Override for /GAME : Must be 0 to enable /GAME signal
- **SPEEDBIAS** 1/2/3.5MHz CPU speed fine adjustment

MEGA65 CPU MATH UNIT REGISTERS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D770	55152	MULTINA							
D771	55153	MULTINA							
D772	55154	MULTINA							
D773	55155	–							MULTINA
D774	55156	MULTINB							
D775	55157	MULTINB							
D776	55158	–						MULTINB	
D778	55160	MULTOUT							
D779	55161	MULTOUT							
D77A	55162	MULTOUT							
D77B	55163	MULTOUT							
D77C	55164	MULTOUT							
D77D	55165	MULTOUT							
D77E	55166	MULTOUT							
D780	55168	MATHIN0							
D781	55169	MATHIN0							
D782	55170	MATHIN0							
D783	55171	MATHIN0							
D784	55172	MATHIN1							
D785	55173	MATHIN1							
D786	55174	MATHIN1							
D787	55175	MATHIN1							
D788	55176	MATHIN2							
D789	55177	MATHIN2							
D78A	55178	MATHIN2							
D78B	55179	MATHIN2							
D78C	55180	MATHIN3							
D78D	55181	MATHIN3							
D78E	55182	MATHIN3							
D78F	55183	MATHIN3							
D790	55184	MATHIN4							
D791	55185	MATHIN4							
D792	55186	MATHIN4							
D793	55187	MATHIN4							
D794	55188	MATHIN5							
D795	55189	MATHIN5							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D796	55190	MATHIN5							
D797	55191	MATHIN5							
D798	55192	MATHIN6							
D799	55193	MATHIN6							
D79A	55194	MATHIN6							
D79B	55195	MATHIN6							
D79C	55196	MATHIN7							
D79D	55197	MATHIN7							
D79E	55198	MATHIN7							
D79F	55199	MATHIN7							
D7A0	55200	MATHIN8							
D7A1	55201	MATHIN8							
D7A2	55202	MATHIN8							
D7A3	55203	MATHIN8							
D7A4	55204	MATHIN9							
D7A5	55205	MATHIN9							
D7A6	55206	MATHIN9							
D7A7	55207	MATHIN9							
D7A8	55208	MATHIN10							
D7A9	55209	MATHIN10							
D7AA	55210	MATHIN10							
D7AB	55211	MATHIN10							
D7AC	55212	MATHIN11							
D7AD	55213	MATHIN11							
D7AE	55214	MATHIN11							
D7AF	55215	MATHIN11							
D7B0	55216	MATHIN12							
D7B1	55217	MATHIN12							
D7B2	55218	MATHIN12							
D7B3	55219	MATHIN12							
D7B4	55220	MATHIN13							
D7B5	55221	MATHIN13							
D7B6	55222	MATHIN13							
D7B7	55223	MATHIN13							
D7B8	55224	MATHIN14							
D7B9	55225	MATHIN14							
D7BA	55226	MATHIN14							
D7BB	55227	MATHIN14							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D7BC	55228	MATHIN15							
D7BD	55229	MATHIN15							
D7BE	55230	MATHIN15							
D7BF	55231	MATHIN15							
D7C0	55232	UNIT0INB				UNIT0INA			
D7C1	55233	UNIT1INB				UNIT1INA			
D7C2	55234	UNIT2INB				UNIT2INA			
D7C3	55235	UNIT3INB				UNIT3INA			
D7C4	55236	UNIT4INB				UNIT4INA			
D7C5	55237	UNIT5INB				UNIT5INA			
D7C6	55238	UNIT6INB				UNIT6INA			
D7C7	55239	UNIT7INB				UNIT7INA			
D7C8	55240	UNIT8INB				UNIT8INA			
D7C9	55241	UNIT9INB				UNIT9INA			
D7CA	55242	UNIT10INB				UNIT10INA			
D7CB	55243	UNIT11INB				UNIT11INA			
D7CC	55244	UNIT12INB				UNIT12INA			
D7CD	55245	UNIT13INB				UNIT13INA			
D7CE	55246	UNIT14INB				UNIT14INA			
D7CF	55247	UNIT15INB				UNIT15INA			
D7D0	55248	-				UNIT0OUT			
D7D1	55249	-				UNIT1OUT			
D7D2	55250	-				UNIT2OUT			
D7D3	55251	-				UNIT3OUT			
D7D4	55252	-				UNIT4OUT			
D7D5	55253	-				UNIT5OUT			
D7D6	55254	-				UNIT6OUT			
D7D7	55255	-				UNIT7OUT			
D7D8	55256	-				UNIT8OUT			
D7D9	55257	-				UNIT9OUT			
D7DA	55258	-				UNIT10OUT			
D7DB	55259	-				UNIT11OUT			
D7DC	55260	-				UNIT12OUT			
D7DD	55261	-				UNIT13OUT			
D7DE	55262	-				UNIT14OUT			
D7DF	55263	-				UNIT15OUT			
D7E0	55264	LATCHINT							
D7E1	55265	-						CALCEN	WREN

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D7E2	55266	RESERVED							
D7E3	55267	RESERVED							

- **CALCEN** Enable committing of output values from math units back to math registers (clearing effectively pauses iterative formulae)
- **LATCHINT** Latch interval for latched outputs (in CPU cycles)
- **MATHIN0** Math unit 32-bit input 0
- **MATHIN1** Math unit 32-bit input 1
- **MATHIN10** Math unit 32-bit input 10
- **MATHIN11** Math unit 32-bit input 11
- **MATHIN12** Math unit 32-bit input 12
- **MATHIN13** Math unit 32-bit input 13
- **MATHIN14** Math unit 32-bit input 14
- **MATHIN15** Math unit 32-bit input 15
- **MATHIN2** Math unit 32-bit input 2
- **MATHIN3** Math unit 32-bit input 3
- **MATHIN4** Math unit 32-bit input 4
- **MATHIN5** Math unit 32-bit input 5
- **MATHIN6** Math unit 32-bit input 6
- **MATHIN7** Math unit 32-bit input 7
- **MATHIN8** Math unit 32-bit input 8
- **MATHIN9** Math unit 32-bit input 9
- **MULTINA** Multiplier input A (25 bit)
- **MULTINB** Multiplier input A (18 bit)
- **MULTOUT** 48-bit output of MULTINA $\times$ MULTINB
- **RESERVED** Reserved
- **UNIT0INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 0.

- **UNIT0INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 0.
- **UNIT0OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 0
- **UNIT10INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 10.
- **UNIT10INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 10.
- **UNIT10OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit A
- **UNIT11INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 11.
- **UNIT11INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 11.
- **UNIT11OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit B
- **UNIT12INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 12.
- **UNIT12INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 12.
- **UNIT12OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit C
- **UNIT13INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 13.
- **UNIT13INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 13.
- **UNIT13OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit D
- **UNIT14INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 14.
- **UNIT14INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 14.
- **UNIT14OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit E

- **UNIT15INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 15.
- **UNIT15INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 15.
- **UNIT15OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit F
- **UNIT1INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 1.
- **UNIT1INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 1.
- **UNIT1OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 1
- **UNIT2INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 2.
- **UNIT2INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 2.
- **UNIT2OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 2
- **UNIT3INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 3.
- **UNIT3INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 3.
- **UNIT3OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 3
- **UNIT4INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 4.
- **UNIT4INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 4.
- **UNIT4OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 4
- **UNIT5INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 5.
- **UNIT5INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 5.

- **UNIT5OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 5
- **UNIT6INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 6.
- **UNIT6INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 6.
- **UNIT6OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 6
- **UNIT7INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 7.
- **UNIT7INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 7.
- **UNIT7OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 7
- **UNIT8INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 8.
- **UNIT8INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 8.
- **UNIT8OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 8
- **UNIT9INA** Select which of the 16 32-bit math registers is input A for Math Function Unit 9.
- **UNIT9INB** Select which of the 16 32-bit math registers is input B for Math Function Unit 9.
- **UNIT9OUT** Select which of the 16 32-bit math registers receives the output of Math Function Unit 9
- **WREN** Enable setting of math registers (must normally be set)

MEGA65 HYPERVISOR MODE

Reset

On power-up or reset, the MEGA65 starts up in hypervisor mode, and expects to find a program in the 16KiB hypervisor memory, and begins executing instructions at address $8100. Normally a JMP instruction will be located at this address, that will jump into a reset routine. That is, the 45GS02 does not use the normal 6502 reset vector. It's function is emulated by the Hyppo hypervisor program, which fetches the address from the 6502 reset vector in the loaded client operating system when exiting hypervisor mode.

The hypervisor memory is automatically mapped on reset to $8000 - $BFFF. This special memory is not able to mapped or in anyway accessed, except when in hypervisor mode. This includes from the serial monitor/debugger interface. This is to protect it from accidental or malicious access from a guest operating system.

Entering / Exiting Hypervisor Mode

Entering the Hypervisor occurs whenever any of the following events occurs:

- **Power-on** When the MEGA65 is first powered on.
- **Reset** If the reset line is lowered, or a watch-dog triggered reset occurs.
- **SYSCALL register accessed** The registers $D640 - $D67F in the MEGA65 IO context trigger SYSCALLs when accessed. This is intended to be the mechanism by which a client operating system or process requests the attention of the hypervisor or operating system.
- **Page Fault** On MEGA65s that feature virtual memory, a page fault will cause a trap to hypervisor mode.
- **Certain keyboard events** Pressing the RESTORE key for >0.5 seconds, or the ALT + TAB key combination traps to the hypervisor. Typically the first is used to launch the freeze menu an the second to toggle the display of debug interface.
- **Accessing virtualised IO devices** For example, if the F011 (internal 3.5" disk drive controller) has been virtualised, then attempting to read or write sectors using this device will cause traps to the hypervisor.
- **Executing an instruction that would lock up the CPU** A number of undocumented opcodes on the 6502 will cause the CPU to lockup. On the MEGA65, instead of locking up, the computer will trap to the hypervisor. This could be used to implement alternative instruction behaviours, or simply to tell the user that something bad has happened.
- **Certain special events** Some devices can generate hypervisor-level interrupts. These are implemented as traps to the hypervisor.

The 45GS02 handles all of these in a similar manner internally:

1. The SYSCALL or trap address is calculated, based on the event.
2. The contents of all CPU registers are saved into the virtualisation control registers.

3. The hypervisor mode memory layout is activated, the CPU decimal flag and special purpose registers are all set to appropriate values. The contents of the A,X,Y and Z and most other CPU flags are preserved, so that they can be accessed from the hypervisor's SYSCALL/trap handler routine, without having to load them, thus saving a few cycles for each call.
4. The hypervisor-mode flag is asserted, and the programme counter (PC) register is set to the computed address.

All of the above happens in one CPU cycle, i.e., in 25 nano-seconds. Returning from a SYSCALL or trap consists simply of writing to $D67F, which requires 125 nano-seconds, for a total overhead of 150 nano-seconds. This gives the MEGA65 SYSCALL performance rivaling – even beating – even the fastest modern computers, where the system call latency is typically hundreds to tens of thousands of cycles [1].

Hypervisor Memory Layout

The hypervisor memory is 16KiB in size. The first 512 bytes are reserved for SYSCALL and system trap entry points, with four bytes for each. For example, the reset entry point is at $8100 - $8100 + 3 = $8100 - $8103. This allows 4 bytes for an instruction, typically a JMP instruction, followed by a NOP to pad it to 4 bytes.

The full list of SYSCALLs and traps is:

HEX	**DEC**	**Name**	**Description**
8000	32768	SYSCALL00	Syscall 0 entry point
8004	32772	SYSCALL01	Syscall 1 entry point
8008	32776	SYSCALL02	Syscall 2 entry point
800C	32780	SYSCALL03	Syscall 3 entry point
8010	32784	SYSCALL04	Syscall 4 entry point
8014	32788	SYSCALL05	Syscall 5 entry point
8018	32792	SYSCALL06	Syscall 6 entry point
801C	32796	SYSCALL07	Syscall 7 entry point
8020	32800	SYSCALL08	Syscall 8 entry point
8024	32804	SYSCALL09	Syscall 9 entry point
8028	32808	SYSCALL0A	Syscall 10 entry point
802C	32812	SYSCALL0B	Syscall 11 entry point
8030	32816	SYSCALL0C	Syscall 12 entry point
8034	32820	SYSCALL0D	Syscall 13 entry point
8038	32824	SYSCALL0E	Syscall 14 entry point
803C	32828	SYSCALL0F	Syscall 15 entry point
8040	32832	SYSCALL10	Syscall 16 entry point

continued ...

...continued

HEX	DEC	Name	Description
8044	32836	SECURENTR	Enter secure container trap entry point
8048	32840	SECUREXIT	Leave secure container trap entry point.
804C	32844	SYSCALL13	Syscall 19 entry point
8050	32848	SYSCALL14	Syscall 20 entry point
8054	32852	SYSCALL15	Syscall 21 entry point
8058	32856	SYSCALL16	Syscall 22 entry point
805C	32860	SYSCALL17	Syscall 23 entry point
8060	32864	SYSCALL18	Syscall 24 entry point
8064	32868	SYSCALL19	Syscall 25 entry point
8068	32872	SYSCALL1A	Syscall 26 entry point
806C	32876	SYSCALL1B	Syscall 27 entry point
8070	32880	SYSCALL1C	Syscall 28 entry point
8074	32884	SYSCALL1D	Syscall 29 entry point
8078	32888	SYSCALL1E	Syscall 30 entry point
807C	32892	SYSCALL1F	Syscall 31 entry point
8080	32896	SYSCALL20	Syscall 32 entry point
8084	32900	SYSCALL21	Syscall 33 entry point
8088	32904	SYSCALL22	Syscall 34 entry point
808C	32908	SYSCALL23	Syscall 35 entry point
8090	32912	SYSCALL24	Syscall 36 entry point
8094	32916	SYSCALL25	Syscall 37 entry point
8098	32920	SYSCALL26	Syscall 38 entry point
809C	32924	SYSCALL27	Syscall 39 entry point
80A0	32928	SYSCALL28	Syscall 40 entry point
80A4	32932	SYSCALL29	Syscall 41 entry point
80A8	32936	SYSCALL2A	Syscall 42 entry point
80AC	32940	SYSCALL2B	Syscall 43 entry point
80B0	32944	SYSCALL2C	Syscall 44 entry point
80B4	32948	SYSCALL2D	Syscall 45 entry point
80B8	32952	SYSCALL2E	Syscall 46 entry point
80BC	32956	SYSCALL2F	Syscall 47 entry point
80C0	32960	SYSCALL30	Syscall 48 entry point
80C4	32964	SYSCALL31	Syscall 49 entry point
80C8	32968	SYSCALL32	Syscall 50 entry point
80CC	32972	SYSCALL33	Syscall 51 entry point
80D0	32976	SYSCALL34	Syscall 52 entry point
80D4	32980	SYSCALL35	Syscall 53 entry point
80D8	32984	SYSCALL36	Syscall 54 entry point

continued ...

...continued

HEX	DEC	Name	Description
80DC	32988	SYSCALL37	Syscall 55 entry point
80E0	32992	SYSCALL38	Syscall 56 entry point
80E4	32996	SYSCALL39	Syscall 57 entry point
80E8	33000	SYSCALL3A	Syscall 58 entry point
80EC	33004	SYSCALL3B	Syscall 59 entry point
80F0	33008	SYSCALL3C	Syscall 60 entry point
80F4	33012	SYSCALL3D	Syscall 61 entry point
80F8	33016	SYSCALL3E	Syscall 62 entry point
80FC	33020	SYSCALL3F	Syscall 63 entry point
8100	33024	RESET	Power-on/reset entry point
8104	33028	PAGFAULT	Page fault entry point (not currently used)
8108	33032	RESTORKEY	Restore-key long press trap entry point
810C	33036	ALTTABKEY	ALT+TAB trap entry point
8110	33040	VF011RD	F011 virtualised disk read trap entry point
8114	33044	VF011WR	F011 virtualised disk write trap entry point
8118	33048	BREAKPT	CPU breakpoint encountered
811C - 81FB	33048 - 33275	RESERVED	Reserved traps point entry
81FC	33276	CPUKIL	KIL instruction in 6502-mode trap entry point

The remainder of the 16KiB hypervisor memory is available for use by the programmer, but xwill typically use the last 512 bytes for the stack and zero-page, giving an overall memory map as follows:

HEX	DEC	Description
8000 - 81FF	32768 - 33279	SYSCALL and trap entry points
8200 - BDFF	33280 - 48639	Available for hypervisor or operating system program
8E00 - BEFF	48640 - 48895	Processor stack for hypervisor or operating system

continued ...

…continued

HEX	DEC	Description
8F00 - BFFF	48896 - 49151	Processor zero-page storage for hypervisor or operating system

The stack is used for holding the return address of function calls. The zero-page storage is typically used for holding variables and other short-term storage, as is customary on the 6502.

Hypervisor Virtualisation Control Registers

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D640	54848	REGA							
D641	54849	REGX							
D643	54851	REGZ							
D644	54852	REGB							
D646	54854	SPH							
D647	54855	PFLAGS							
D648	54856	PCL							
D649	54857	PCH							
D64A	54858	MAPLO							
D64B	54859	MAPLO							
D64C	54860	MAPHI							
D64D	54861	MAPHI							
D64E	54862	MAPLOMB							
D64F	54863	MAPHIMB							
D650	54864	PORT00							
D651	54865	PORT01							
D652	54866	-					EXSID	VICMODE	
D653	54867	DMASRCMB							
D654	54868	DMADSTMB							
D655	54869	DMALADDR							
D656	54870	DMALADDR							
D657	54871	DMALADDR							
D658	54872	DMALADDR							
D659	54873	-							VFLOP
D670	54896	GEORAMBASE							
D671	54897	GEORAMMASK							
D672	54898	-	MATRIXEN	-					

continued …

…continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D67C	54908	UARTDATA							
D67D	54909	WATCHDOG							
D67E	54910	HICKED							
D67F	54911	ENTEREXIT							

- **ASCFAST** Hypervisor enable ASC/DIN CAPS LOCK key to enable/disable CPU slow-down in C64/C128/C65 modes
- **CARTEN** Hypervisor enable /EXROM and /GAME from cartridge
- **CPUFAST** Hypervisor force CPU to 48MHz for userland (userland can override via POKE0)
- **DMADSTMB** Hypervisor DMAgic destination MB
- **DMALADDR** Hypervisor DMAGic list address bits 0-7
- **DMASRCMB** Hypervisor DMAgic source MB
- **ENTEREXIT** Writing trigger return from hypervisor
- **EXSID** 0=Use internal SIDs, 1=Use external(1) SIDs
- **F4502** Hypervisor force CPU to 4502 personality, even in C64 IO mode.
- **GEORAMBASE** Hypervisor GeoRAM base address (x MB)
- **GEORAMMASK** Hypervisor GeoRAM address mask (applied to GeoRAM block register)
- **HICKED** Hypervisor already-upgraded bit (writing sets permanently)
- **JMP32EN** Hypervisor enable 32-bit JMP/JSR etc
- **MAPHI** Hypervisor MAPHI register storage (high bits)
- **MAPHIMB** Hypervisor MAPHI mega-byte number register storage
- **MAPLO** Hypervisor MAPLO register storage (high bits)
- **MAPLOMB** Hypervisor MAPLO mega-byte number register storage
- **MATRIXEN** Enable composited Matrix Mode, and disable UART access to serial monitor.
- **PCH** Hypervisor PC-high register storage
- **PCL** Hypervisor PC-low register storage
- **PFLAGS** Hypervisor P register storage

- **PIRQ** Hypervisor flag to indicate if an IRQ is pending on exit from the hypervisor / set 1 to force IRQ/NMI deferal for 1,024 cycles on exit from hypervisor.
- **PNMI** Hypervisor flag to indicate if an NMI is pending on exit from the hypervisor.
- **PORT00** Hypervisor CPU port $00 value
- **PORT01** Hypervisor CPU port $01 value
- **REGA** Hypervisor A register storage
- **REGB** Hypervisor B register storage
- **REGX** Hypervisor X register storage
- **REGZ** Hypervisor Z register storage
- **ROMPROT** Hypervisor write protect C65 ROM $20000-$3FFFF
- **SPH** Hypervisor SPH register storage
- **UARTDATA** (write) Hypervisor write serial output to UART monitor
- **VFLOP** 1=Virtualise SD/Floppy access (usually for access via serial debugger interface)
- **VICMODE** VIC-II/VIC-III/VIC-IV mode select
- **WATCHDOG** Hypervisor watchdog register: writing any value clears the watch dog

Programming for Hypervisor Mode

The easiest way to write a program for Hypervisor Mode on the MEGA65 is to use KickC, which is a special version of C made for writing programs for 6502-class processors. The following example programs are from KickC's supplied examples. KickC produces very efficient code, and directly supports the MEGA65's hypervisor mode quite easily through the use of a linker definition file with the following contents:

```
.file [name="%O.bin", type="bin", segments="XMega65Bin"]
.segmentdef XMega65Bin [segments="Syscall, Code, Data, Stack, Zeropage"]
.segmentdef Syscall [start=$8000, max=$81ff]
.segmentdef Code [start=$8200, min=$8200, max=$bdff]
.segmentdef Data [startAfter="Code", min=$8200, max=$bdff]
.segmentdef Stack [min=$be00, max=$beff, fill]
.segmentdef Zeropage [min=$bf00, max=$bfff, fill]
```

This file instructs KickC's assembler to create a 16KiB file with the 512 byte SYSCALL/trap entry point region at the start, followed by code and data areas, and then the stack and zero-page areas. It enforces the size and location of these fields, and will give an error during compilation if anything is too big to fit.

With this file in place, you can then create a KickC source file that provides data structures for the SYSCALL/trap table, e.g.:

```
// XMega65 Kernal Development Template
// Each function of the kernal is a no-args function
// The functions are placed in the SYSCALLS table surrounded by JMP and NOP

import "string"

// Use a linker definition file (put the previous listing into that file)
#pragma link("mega65hyper.ld")

// Some definitions of addresses and special values that this program uses
const char* RASTER = 0xd012;
const char* VIC_MEMORY = 0xd018;
const char* SCREEN = 0x0400;
const char* BGCOL = 0xd021;
const char* COLS = 0xd800;
const char BLACK = 0;
const char WHITE = 1;

// Some text to display
char[] MESSAGE = "hello world!";

void main() {
    // Initialize screen memory, and select correct font
    *VIC_MEMORY = 0x14;
    // Fill the screen with spaces
    memset(SCREEN, ' ', 40*25);
    // Set the colour of every character on the screen to white
    memset(COLS, WHITE, 40*25);
    // Print the "hello world!" message
    char* sc = SCREEN+40;  // Display it one line down on the screen
    char* msg = MESSAGE; // The messag to display
    // A simple copy routine to copy the string
    while(*msg) {
        *sc++ = *msg++;
    }
    // Loop forever showing two white lines as raster bars
    while(true) {
        if(*RASTER==54 || *RASTER==66) {
            *BGCOL = WHITE;
        } else {
            *BGCOL = BLACK;
        }
    }
}

// Here are a couple sample SYSCALL handlers that just display a character on the screen
void syscall1() {
    *(SCREEN+79) = '>';
}

void syscall2() {
    *(SCREEN+78) = '<';
}

// Now we select the SYSCALL segment to hold the SYSCALL/trap entry point table.
#pragma data_seg(Syscall)

// The structure of each entry point is JMP <handler address> + NOP.
// We have a char (xjmp) to hold the opcode for the JMP instruction,
// and then put the address of the SYSCALL/trap handler in the next
// two points as a pointer, and end with the NOP instruction opcode.
struct SysCall {
    char xjmp;         // Holds $4C, the JMP $nnnn opcode
    void()* syscall;   // Holds handler address, will be the target of the JMP
    char xnop;         // Holds $EA, the NOP opcode
};

// To save writing 0x4C and 0xEA all the time, we define them as constants
const char JMP = 0x4c;
const char NOP = 0xea;

// Now we can have a nice table of up to 64 SYSCALL handlers expressed
// in a fairly readable and easy format.
// Each line is an instance of the struct SysCall from above, with the JMP
// opcode value, the address of the handler routine and the NOP opcode value.
export struct SysCall[] SYSCALLS = {
    { JMP, &syscall1, NOP },
    { JMP, &syscall2, NOP }
    };

// In this example we had only two SYSCALLs defined, so rather than having
// another 62 lines, we can just ask KickC to make the TRAP table begin
// at the next multiple of $100, i.e., at $8100.
export align(0x100) struct SysCall[] SYSCALL←RESET = {
    { JMP, &main, NOP }
};
```

If you save the first listing into a file called mega65hyper.ld, and the second into a file called mega65hyper.kc, you can then compile them using KickC with a command like:

```
  kickc -a mega65hyper
```

It will then produce a file called mega65hyper.bin, which you can then try out on your MEGA65, or run in the Xmega65 emulator with a command like:

```
  xmega65 -kickup mega65hyper.bin
```

APPENDIX B

VIC-IV Video Interface Controller

- **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-IV).
- 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 character modes**.
- Support for upto **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 upto 256 colours**. n

- 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 backrounds 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 pseuso-sprites**, similar to "bobs" on Amiga(tm) computers, but with the advantage that they are rended in the display pipeline, and thus do not need to be undrawn and redrawn to animate them.
- **Multiple 8-bit colour playfields** 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 disctinctiveness 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 capabilitie 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 which mode in which the VIC-IV will operate: It is always in VIC-IV mode. However, for backwards compatibility with softwre, 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 kock 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 visibiel (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 IO 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 53295, SO WE TEST A DIFFERENT WAY
10 IF PEEK(53272) AND 32 THEN GOTO 65
20 POKE53248,1:POKE53295,71:POKE53295,83
```

```
30 POKE53248+256,0:IFPEEK(53248)=1THENPRINT"VIC-IV PRESENT":END
40 POKE53248,1:POKE53295,165:POKE53295,150
50 POKE53248+256,0:IFPEEK(53248)=1THENPRINT"VIC-III PRESENT":END
60 PRINT "VIC-II PRESENT": END

65 REM WE ASSUME WE HAVE A C65 HERE
70 V1=PEEK(53248+80):V2=PEEK(53248+80):V3=PEEK(53248+80)
80 IF V1<>V2 OR V1<>V3 OR V2<>V3 THEN PRINT "VIC-IV PRESENT":END
90 GOTO 40
```

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 similarily be reasonably assumed to indicate the presence of a C65.

VIDEO OUTPUT FORMATS, TIMING AND COMPATABILITY

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 a connector for the digital video that accepts most cables intended for connecting HDMI(tm) compatible devices. 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 compatability, this display format has several effects that do not cause problems for most programs, but can cause some differences in behaviour:

1. Because 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 1MHz timing impulses 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.

XXX - Diagrams showing VIC-IV PAL / NTSC mode frame timings

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.

To determine whether the MEGA65 is operating in PAL or NTSC, 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 IFPEEK(53272)<32THENPOKE53295,ASC("G"):POKE53295,ASC("S"):REM ENABLE C65+MEGA65 IO
20 NTSC=PEEK(53359)AND128
30 IFNTSCTHENPRINT"MEGA65 IS IN NTSC MODE"
40 IFNTSC=0THENPRINT"MEGA65 IS IN PAL MODE"
50 INPUT"SWITCH MODES (Y/N)? ",A$
60 IFA$="Y"THENPOKE53359,PEEK(53359)+128-NTSC
70 NTSC=PEEK(53359)AND128
80 IFNTSCTHENPRINT"MEGA65 IS NOW IN NTSC MODE"
90 IFNTSC=0THENPRINT"MEGA65 IS NOW IN PAL MODE"
```

MEMORY INTERFACE

The VIC-IV supports upto 64KB of colour RAM and, in principle, 16MB of direct access RAM for video data. However in typical installations 32KB of colour RAM and either 384KB of addressable RAM is present. In MEGA65 systems, the second 128KB of RAM is typically used to hold a C65-compatible ROM, leaving 256KB available, unless software is written to avoid the need to use C65 ROM routines, in which case all 384KB can be used.

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.

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:

```
IFPEEK(53272)<32THENPOKE53295,ASC("G"):POKE53295,ASC("S"):REM ENABLE C65+MEGA65 IO
POKE53344+0,69:POKE53344+1,35:POKE53344+2,1
```

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 accessable 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 AND binary operator only works with arguments between 0 and 65,535. Therefore we first subtract 4×65,536 = 262,144 from the address (the 4 is determined by calculating INT(266752/65536)), before we use the AND operator to compute the lower part of the address:

```
IFPEEK(53272)<32THENPOKE53295,ASC("G"):POKE53295,ASC("S"):REM ENABLE C65+MEGA65 IO
POKE53352,(266752-INT(266752/65536)*65536)AND255
POKE53353,INT((266752-INT(266752/65536)*65536)/256)
POKE53354,INT(266752/65536)
```

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:

```
IFPEEK(53272)<32THENPOKE53295,ASC("G"):POKE53295,ASC("S"):REM ENABLE C65+MEGA65 IO
POKE53348,4000 AND 255
POKE53349,INT(4000/256)
```

Relocating Sprite Pointers and Images

The location of the sprite pointers can also be moved, and sprites can be made to have there 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 arbitary location in the first 8MB of RAM. 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 byts 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 explcitly via the SPRPTRADR registers. For example, to position the sprite pointers at location 800 – 815, you could use something like the following. 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:

```
IFPEEK(53272)<32THENPOKE53295,ASC("G"):POKE53295,ASC("S"):REM ENABLE C65+MEGA65 IO
POKE53356,(800-INT(800/65536)*65536) AND 255
POKE53357,INT(800/256)AND255
POKE53358,(PEEK(53358)AND128)+INT(800/65536)
```

The location of each sprite image remains a multiple of 64 bytes, thus allowing for upto 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 $\times$ the pointer value. For example, to have the data for a sprite at $C000 (49152 decimal), this would be sprite location 768, because 49152 divided by 64 = 768. We then need to split 768 into high and low bytes, to set the two pointer byes: 768 = 256$\times$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:

```
POKE2040,INT(768/256)
POKE2041,768-256*INT(768/256)
```

HOT REGISTERS

Because of the availability of precise vernier registers to set a wide range of video parameters directly, $D011 (53265 decimal), $D016 (53272 decimal) and other VIC-II and VIC-III video mode registers are implemented as virtual registers: by default, writing to any of these results in computed consistent values being applied to all of the relevant vernier registers. This means that writing to any of these virtual registers will reset the video mode. Thus some care has to be taken when using new VIC-IV features to not touch any of the "hot" VIC-II and VIC-III registers.

The "hot" registers to be careful with are:

$D011, $D016, $D018, $D031 (53265, 53272, 53274 and 53287 decimal) and the VIC-II bank bits of $DD00 (56576 decimal).

If you write to any of those, various VIC-IV registers will need to be re-written with the values you wish to maintain.

This "hot" register behaviour is intended primarily for legacy software. It can be disabled by clearing the HOTREG signal (bit 7 of $D05D, 53341 decimal).

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 bitplaner 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 on-screen 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 chipset to include so-called "chunky" modes, rather than just bitplanar 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 Super-Extended Attribute Mode 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 nybl 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, for example, **??**. Super-Extended Attribute mode doubles the number of bits per used to describe each character 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. The default is as follows:

Bit(s)	**Function**
Screen RAM byte 0	Lower 8 bits of character number, the same as the VIC-II and VIC-III

continued ...

...continued

Bit(s)	**Function**
Screen RAM byte 1, bits 0 - 4	Upper 5 bits of character number, allowing addressing of 8,192 unique characters
Screen RAM byte 1, bits 5 - 7	Trim pixels from right side of character (bits 0 - 2)
Colour RAM byte 0, bit 7	Vertically flip the character
Colour RAM byte 0, bit 6	Horizontally flip the character
Colour RAM byte 0, bit 5	Alpha blend mode (leave 0, discussed later)
Colour RAM byte 0, bit 4	GOTO X (allows repositioning of characters along a raster via the Raster-Rewrite Buffer, discussed later)
Colour RAM byte 0, bits 3	If set, Full-Colour characters use 4 bits per pixel and are 16 pixels wide (less right side trim bits), instead of using 8 bits per pixel, and being the usual 8 pixels wide
Colour RAM byte 0, bits 2	Trim pixels from right side of character (bit 3)
Colour RAM byte 0, bits 0 - 1	Number of pixels to trim from top or bottom of character
Colour RAM byte 1, bits 0 - 3	Low 4 bits of colour of character
Colour RAM byte 1, bit 4	Hardware blink of character (if VIC-III extended attributes are enabled)
Colour RAM byte 1, bit 5	Hardware reverse video enable of character (if VIC-III extended attributes are enabled)*
Colour RAM byte 1, bit 6	Hardware bold attribute of character (if VIC-III extended attributes are enabled)*
Colour RAM byte 1, bit 7	Hardware underlining of character (if VIC-III extended attributes are enabled)

* 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.

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.

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

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 upto 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 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 Example program

Raster-Rewrite 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 limted 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.

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 playfields on other computers, such as the Amiga, but without the disadvantagese of bitplanes.

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 8×21 = 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 8×255 = 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 $64 = 16$ pixels wide.
3. Tiled mode, where the sprite is drawn repeatedly until the end of the raster line.

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).

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).

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 phsyical 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 approprate 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: $spritenumber \times 16 + nyblvalue$, where $spritenumber$ is the number of the sprite (from 0 to 7), and $nyblvalue$ 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.

VIC-II / C64 REGISTERS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D000	53248	S0X							
D001	53249	S0Y							
D002	53250	S1X							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D003	53251	S1Y							
D004	53252	S2X							
D005	53253	S2Y							
D006	53254	S3X							
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	RC	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	-					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							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D029	53289	SPR2COL							
D02A	53290	SPR3COL							
D02B	53291	SPR4COL							
D02C	53292	SPR5COL							
D02D	53293	SPR6COL							
D02E	53294	SPR7COL							
D030	53296	-							C128FAST

- **BLNK** disable 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 ($\times$ 1KiB)
- **CSEL** 38/40 column select
- **ECM** extended background mode
- **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 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
- **S0X** sprite 0 horizontal position
- **S0Y** sprite 0 vertical position
- **S1X** sprite 1 horizontal position
- **S1Y** sprite 1 vertical position
- **S2X** sprite 2 horizontal position
- **S2Y** sprite 2 vertical position
- **S3X** sprite 3 horizontal position
- **S3Y** sprite 3 vertical position
- **S4X** sprite 4 horizontal position
- **S4Y** sprite 4 vertical position
- **S5X** sprite 5 horizontal position
- **S5Y** sprite 5 vertical position
- **S6X** sprite 6 horizontal position
- **S6Y** sprite 6 vertical position
- **S7X** sprite 7 horizontal position
- **S7Y** sprite 7 vertical position
- **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
- **SPR0COL** sprite 0 colour / 16-colour sprite transparency colour (lower nybl)
- **SPR1COL** sprite 1 colour / 16-colour sprite transparency colour (lower nybl)
- **SPR2COL** sprite 2 colour / 16-colour sprite transparency colour (lower nybl)
- **SPR3COL** sprite 3 colour / 16-colour sprite transparency colour (lower nybl)

- **SPR4COL** sprite 4 colour / 16-colour sprite transparency colour (lower nybl)
- **SPR5COL** sprite 5 colour / 16-colour sprite transparency colour (lower nybl)
- **SPR6COL** sprite 6 colour / 16-colour sprite transparency colour (lower nybl)
- **SPR7COL** sprite 7 colour / 16-colour sprite transparency colour (lower nybl)
- **SPRMC0** Sprite multi-colour 0 (always 256 colour)
- **SPRMC1** Sprite multi-colour 1 (always 256 colour)
- **SSC** sprite/sprite collision indicate bits
- **SXMSB** sprite horizontal position MSBs
- **VS** screen address ($\times$ 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	ADDR	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							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D03C	53308	BPX							
D03D	53309	BPY							
D03E	53310	HPOS							
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							

- **ADDR** Enable extended attributes and 8 bit colour entries
- **B0ADEVN** – Bitplane 0 address, even lines
- **B0ADODD** – Bitplane 0 address, odd lines
- **B0PIX** Display Address Translater (DAT) Bitplane 0 port
- **B1ADEVN** – Bitplane 1 address, even lines
- **B1ADODD** – Bitplane 1 address, odd lines
- **B1PIX** Display Address Translater (DAT) Bitplane 1 port
- **B2ADEVN** – Bitplane 2 address, even lines
- **B2ADODD** – Bitplane 2 address, odd lines
- **B2PIX** Display Address Translater (DAT) Bitplane 2 port
- **B3ADEVN** – Bitplane 3 address, even lines
- **B3ADODD** – Bitplane 3 address, odd lines

- **B3PIX** Display Address Translater (DAT) Bitplane 3 port
- **B4ADEVN** – Bitplane 4 address, even lines
- **B4ADODD** – Bitplane 4 address, odd lines
- **B4PIX** Display Address Translater (DAT) Bitplane 4 port
- **B5ADEVN** – Bitplane 5 address, even lines
- **B5ADODD** – Bitplane 5 address, odd lines
- **B5PIX** Display Address Translater (DAT) Bitplane 5 port
- **B6ADEVN** – Bitplane 6 address, even lines
- **B6ADODD** – Bitplane 6 address, odd lines
- **B6PIX** Display Address Translater (DAT) Bitplane 6 port
- **B7ADEVN** – Bitplane 7 address, even lines
- **B7ADODD** – Bitplane 7 address, odd lines
- **B7PIX** Display Address Translater (DAT) Bitplane 7 port
- **BORDERCOL** display border colour (256 colour)
- **BPCOMP** Complement bitplane flags
- **BPM** Bit-Plane Mode
- **BPX** Bitplane X
- **BPY** Bitplane Y
- **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 $A5then$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 video output (not implemented)
- **PAL** Use PALETTE ROM or RAM 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	-		XPOS					
D051	53329	XPOS							
D052	53330	FNRASTER							
D053	53331	FNRST	-				FNRASTER		
D054	53332	ALPHEN	VFAST	PALEMU	SPR640	SMTH	FCLRHI	FCLRLO	CHR16
D055	53333	SPRHGTEN							
D056	53334	SPRHGHT							
D057	53335	SPRX64EN							
D058	53336	CHARSTEP							
D059	53337	CHARSTEP							
D05A	53338	CHRXSCL							
D05B	53339	CHRYSCL							
D05C	53340	SIDBDRWD							
D05D	53341	HOTREG	RSTDELEN	SIDBDRWD					
D05E	53342	CHRCOUNT							
D05F	53343	SPRXSMSBS							
D060	53344	SCRNPTR							
D061	53345	SCRNPTR							
D062	53346	SCRNPTR							
D063	53347	SCRNPTR							
D064	53348	COLPTR							
D065	53349	COLPTR							
D068	53352	CHARPTR							
D069	53353	CHRPTR							
D06A	53354	CHRPTR							
D06B	53355	SPR16EN							
D06C	53356	SPRPTRADR							
D06D	53357	SPRPTRADR							
D06E	53358	SPRPTR16	SPRPTRBNK						
D06F	53359	PALNTSC	-	RASLINE0					
D070	53360	MAPEDPAL		BTPALSEL		SPRPALSEL		ABTPALSEL	
D071	53361	BP16ENS							
D072	53362	VSYNDEL							

continued ...

...continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D073	53363	RASTERHEIGHT				ALPHADELAY			
D074	53364	SPRENALPHA							
D075	53365	SPRALPHAVAL							
D076	53366	SPRENV400							
D077	53367	SRPYMSBS							
D078	53368	SPRYSMSBS							
D079	53369	RSTCOMP							
D07A	53370	FNRSTCMP	RESERVED				RSTCMP		
D07B	53371	RESERVED							
D07C	53372	RESERVED		VSYNCP	HSYNCP	RESERVED			

- **ABTPALSEL** VIC-IV bitmap/text palette bank (alternate palette)
- **ALPHADELAY** Alpha delay for compositor
- **ALPHEN** Alpha compositor enable
- **BBDRPOS** bottom border position
- **BORDERCOL** display border colour (256 colour)
- **BP16ENS** VIC-IV 16-colour bitplane enable flags
- **BTPALSEL** bitmap/text palette bank
- **CHARPTR** Character set precise base address (bits 0 - 7)
- **CHARSTEP** characters per logical text row (LSB)
- **CHR16** enable 16-bit character numbers (two screen bytes per character)
- **CHRCOUNT** Number of characters to display per row
- **CHRPTR** Character set precise base address (bits 15 - 8)
- **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)
- **COLPTR** colour RAM base address (bits 0 - 7)
- **FCLRHI** enable full-colour mode for character numbers >$FF
- **FCLRLO** enable full-colour mode for character numbers <=$FF
- **FNRASTER** Read physical raster position

- **FNRST** Raster compare source (1=VIC-IV fine raster, 0=VIC-II raster)
- **FNRSTCMP** Raster compare is in physical rasters if set, or VIC-II raster if clear
- **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
- **HSYNCP** hsync polarity
- **KEY** Write $47then$53 to enable C65GS/VIC-IV IO registers
- **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)
- **PALEMU** video output pal simulation
- **PALNTSC** NTSC emulation mode (max raster = 262)
- **RASLINE0** first VIC-II raster line
- **RASTERHEIGHT** physical rasters per VIC-II raster (1 to 16)
- **RESERVED**
- **RSTCMP** Raster compare value MSB
- **RSTCOMP** Raster compare value
- **RSTDELEN** Enable raster delay (delays raster counter and interrupts by one line to match output pipeline latency)
- **SCREENCOL** screen colour (256 colour)
- **SCRNPTR** screen RAM precise base address (bits 0 - 7)
- **SIDBDRWD** Width of single side border
- **SMTH** video output horizontal smoothing enable
- **SPR16EN** sprite 16-colour mode enables
- **SPR640** Sprite H640 enable;
- **SPRALPHAVAL** Sprite alpha-blend value
- **SPRBPMEN** Sprite bitplane-modify-mode enables
- **SPRENALPHA** Sprite alpha-blend enable

- **SPRENV400** Sprite V400 enables
- **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)
- **SPRPTRADR** sprite pointer address (bits 7 - 0)
- **SPRPTRBNK** sprite pointer address (bits 22 - 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
- **SPRYSMSBS** Sprite V400 Y position super MSBs
- **SRPYMSBS** Sprite V400 Y position MSBs
- **TBDRPOS** top border position
- **TEXTXPOS** character generator horizontal position
- **TEXTYPOS** Character generator vertical position
- **VFAST** C65GS FAST mode (48MHz)
- **VSYNCP** vsync polarity
- **VSYNDEL** VIC-IV VSYNC delay
- **XPOS** Read horizontal raster scan position

APPENDIX C

6526 Complex Interface Adapter (CIA) Registers

CIA 6526 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	–		TODSEC					
DC0B	56331	TODAMPM	–		TODHOUR				
DC0C	56332	SDR							
DC0D	56333	IR	–		FLG	SP	ALRM	TB	TA
DC0E	56334	TOD50	SPMOD	IMODA	–	RMODA	OMODA	PBONA	STRTA
DC0F	56335	–	IMODB		LOAD	RMODB	OMODB	PBONB	STRTB

- **ALRM** TOD alarm
- **DDRA** Port A DDR
- **DDRB** Port B DDR
- **FLG** FLAG edge detected
- **IMODA** Timer A Timer A tick source
- **IMODB** Timer B Timer A tick source
- **IR** Interrupt flag
- **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
- **TODHOUR** TOD hours
- **TODJIF** TOD tenths of seconds
- **TODSEC** TOD seconds

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DC0F	56335	TODEDIT	-						
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					
DD0A	56586	-		TODSEC					
DD0B	56587	TODAMPM	-		TODHOUR				
DD0C	56588	SDR							

continued ...

…continued

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DD0D	56589	-			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 Timer A tick source
- **IMODB** Timer B Timer A tick source
- **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.

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	TODAMPM	TODHOUR						
DC1C	56348	ALRMJIF							
DC1D	56349	ALRMSEC							
DC1E	56350	ALRMMIN							
DC1F	56351	ALRMAMPM	ALRMHOUR						

- **ALRMAMPM** TOD Alarm AM/PM flag
- **ALRMHOUR** TOD Alarm hours value
- **ALRMJIF** TOD Alarm 10ths of seconds value

- **ALRMMIN** TOD Alarm minutes value
- **ALRMSEC** TOD Alarm seconds value
- **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

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	TODAMPM	TODHOUR						
DD1C	56604	ALRMJIF							
DD1D	56605	ALRMSEC							
DD1E	56606	ALRMMIN							
DD1F	56607	ALRMAMPM	ALRMHOUR						

- **ALRMAMPM** TOD Alarm AM/PM flag
- **ALRMHOUR** TOD Alarm hours value
- **ALRMJIF** TOD Alarm 10ths of seconds value
- **ALRMMIN** TOD Alarm minutes value

- **ALRMSEC** TOD Alarm seconds value
- **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

APPENDIX D

4551 UART, GPIO and Utility Controller

C65 6551 UART REGISTERS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D600	54784	DATA							
D601	54785	–				FRMERR	PTYERR	RXOVRRUN	RXRDY
D602	54786	TXEN	RXEN	SYNCMOD		CHARSZ		PTYEN	PTYEVE
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 IO & MISCELLANEOUS INTERFACE REGISTERS

HEX	DEC	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
D609	54793	-							UFAST
D60B	54795	OSKZEN	OSKZON	PORTF					
D60C	54796	PORTFDDR		PORTFDDR					
D60D	54797	HDSCL	HDSPI	SDBSH	SDCS	SDCLK	SDDATA	RST41	CONN41
D60E	54798	BASHDDR							
D60F	54799	-						KEYUP	KEYLEFT
D610	54800	ASCIIKEY							
D611	54801	-		MSCRL	MALT	MMEGA	MCTRL	MLSHFT	MRSHFT
D612	54802	LJOYB	LJOYA	PHYJOY	PS2JOY	VRTKEY	PHYKEY	PS2KEY	WGTKEY
D615	54805	OSKEN	VIRTKEY1						
D616	54806	OSKALT	VIRTKEY2						
D617	54807	OSKTOP	VIRTKEY3						
D618	54808	KSCNRATE							
D619	54809	OSKXPOS							
D61A	54810	OSKYPOS							
D620	54816	POTAX							
D621	54817	POTAY							
D622	54818	POTBX							
D623	54819	POTBY							

- **ASCIIKEY** Last key press as ASCII (hardware accelerated keyboard scanner). Write to clear event ready for next.
- **BASHDDR** Data Direction Register (DDR) for $D60D bit bashing port.
- **CONN41** Internal 1541 drive connect (1=connect internal 1541 drive to IEC bus)
- **HDSCL** HDMI SPI control interface SCL clock

- **HDSPI** HDMI SPI control interface SDA data line
- **KEYLEFT** Directly read C65 Cursor left key
- **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)
- **MALT** ALT key state (hardware accelerated keyboard scanner).
- **MCTRL** CTRL key state (hardware accelerated keyboard scanner).
- **MLSHFT** Left shift key state (hardware accelerated keyboard scanner).
- **MMEGA** MEGA/C= key state (hardware accelerated keyboard scanner).
- **MRSHFT** Right shift key state (hardware accelerated keyboard scanner).
- **MSCRL** NOSCRL key state (hardware accelerated keyboard scanner).
- **OSKALT** Display alternate on-screen keyboard layout (typically dial pad for MEGA65 telephone)
- **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.
- **OSKXPOS** Set on-screen keyboard X position (x4 640H pixels)
- **OSKYPOS** Set on-screen keyboard Y position (x4 physical pixels)
- **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
- **PHYJOY** Enable physical joystick input
- **PHYKEY** Enable physical keyboard input
- **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.
- **PS2JOY** Enable PS/2 / USB keyboard simulated joystick input
- **PS2KEY** Enable ps2 keyboard/joystick input
- **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
- **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.
- **VRTKEY** Enable virtual/snythetic keyboard input
- **WGTKEY** Enable widget board keyboard/joystick input

APPENDIX E

Reference Tables

- Units of Storage
- Base Conversion

UNITS OF STORAGE

Unit	**Equals**	**Abbreviation**
1 Bit		
1 Nybble	4 Bits	
1 Byte	8 bits	B
1 Kilobyte	1024 B	KB
1 Megabyte	1024 KB 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

APPENDIX F

Using a Nexys4DDR as a MEGA65

- Building your own MEGA65 Compatible Computer
- Power, Jumpers, Switches and Buttons
- Keyboard
- Preparing microSDHC card
- Useful Tips

BUILDING YOUR OWN MEGA65 COMPATIBLE COMPUTER

You can build your own MEGA65-compatible computer by using a Nexys4DDR FPGA development board. This appendix describes the process to setup a Nexys4DDR board for this purpose. The older non-DDR Nexys4 board is also supported, and the instructions are the same, except that you must use a bitstream designed for that board. Using a Nexys4DDR bitstream on a non-DDR Nexys4 board, or vice versa, may cause irrepairable damage to your board, so make sure you have the correct bitstream to suit your board.

DISCLAIMER: M.E.G.A cannot take any responsibility for any damage that may occur to your Nexys4DDR board.

POWER, JUMPERS, SWITCHES AND BUTTONS

The MEGA65 core consumes too much power for the Nexys4DDR board to be powered from a standard USB port. In particular, writing to the SD card might hang or perform odd behaviour. Therefore you should use a 5V power supply. Digilent sell a power supply for the Nexys4DDR board, and we recommend you use this to ensure you avoid the risk of damage to your Nexys4DDR board.

Set the following jumpers on your Nexys4DDR board to the following positions:

- JP1 - USB/SD
- JP2 - SD
- JP3 - Wall

All 16 switches on the lower edge of the board must be set to the off position.

The "CPU RESET" button will reset the MEGA65 when pressed, while the "PROG" button will cause the FPGA itself to reload the MEGA65 core. The main difference between the two is that CPU RESET is faster, and does not clear the contents of memory, while the FPGA button is slower, and does reset the contents of memory.

Two of the five buttons in the cross arrangement can also be used: BTNU acts as though you have pressed the **RESTORE** key, while BTNC will trigger an IRQ, as though the IRQ line had been pulled to ground.

KEYBOARD

Only USB keyboards that lack a USB hub will work with the Nexys4DDR board. Generally extremely cheap keyboards will work, while more expensive keyboards tend to have a USB hub integrated, and will not work. You may need to try several keyboards, before you find one that works.

The keyboard layout is positional rather than logical. This means that keys in similar positions to the keys on a C65 keyboard will have similar function. This relationship assumes that your USB keyboard uses a US keyboard layout.

The **RESTORE** key is mapped to the PAGE UP key.

PREPARING MICROSDHC CARD

The MEGA65 requires an SDHC card of between 4GB and 64GB capacity. Some SDXC cards may work, however, this is not officially supported.

To prepare your SD card, you will need the following two separate files from the MEGA65 web site:

- Bitstream `http://mega65.org/assets/bitstream.7z`
- Support Files `http://mega65.org/assets/fileset.7z`

In addition, you will also need a C65 ROM. There were many different versions created during the development of the Commodore 65, and the MEGA65 can use any of them. However, we recommend you use 911001.bin, as this has the most complete BASIC and DOS implementations.

The steps are:

- Format the SD card in a convenient computer using the FAT32 filesystem. The MEGA65 and Nexys4DDR boards do not understand other file systems, especially the exFAT file system.
- Unzip the bitstream.7z file, and copy the file with name ending in ".bit" onto the SD card.
- Insert the SD card into the SD card slot on the under-side of the Nexys4DDR board.
- Turn on the Nexys4DDR board.
- Enter the Utility Menu by holding the left and right SHIFT keys down on the USB keyboard you have connected to the Nexys4DDR board.
- Enter the FDISK/FORMAT tool by pressing 2 when the option appears on the MEGA65 boot screen.
- Follow the prompts in the FDISK/FORMAT program to again format the SD card for use by the MEGA65. (The FDISK tool will partition your SD card into two partitions and format them. One is type $41 = MEGA65 System Partition, where the save slots, configuration data and other files live. (This partition is invisible in i.e. Win PCs). and the other partition with type $0C = VFAT32, where KERNEL, support files, games, and so on, will be copied to later. (This partition is visible on i.e. Win PCs).
- Once formatting is complete, switch off the Nexys4DDR board and remove the microSDHC card from the Nexys board and put it back into your PC
- Again copy the bitstream back onto the SD card, as well as all the files from the other 7z file downloaded from the MEGA65 website.

- Copy the 911001.bin ROM file onto the SD card, and rename it to MEGA65.ROM
- If you have any .D81 files, copy them onto the SD card. Make sure that they have names that fit the old DOS 8.3 character limit, and are upper case. This restriction will be removed in a future release.
- Remove the SD card and reinsert it into your Nexys4DDR board.
- Power the Nexys4DDR board back on. The MEGA65 should boot within 15 seconds.

Congratulations. Your MEGA65 has been set up and is ready to use.

USEFUL TIPS

The following are some useful tips for getting familiar with the MEGA65:

- Press & hold (or the Commodore key if using a Commodore 64 or 65 keyboard) during boot to start up in C64 mode instead of C65 mode
- Press & hold **RUN STOP** during boot to enter the machine language monitor, instead of starting BASIC.
- Press the **RESTORE** key for approximately 1/2 - 1 second to enter the MEGA65 Freeze Menu. From this menu you have convenient tools to change the CPU speed, switch between PAL & NTSC video mode, change Audio settings, manage freeze-states, select D81 disk images, examine and modify memory of the frozen program, among other features. This is in many ways the heart of the MEGA65, so it is well worth exploring and getting familiar with.
- Press the **RESTORE** for about 2 seconds to reset the MEGA65. This is a temporary feature that will be removed when the MEGA65 desktop computers with built-in reset button become available.
- Type `POKE0,65` in C64 mode to switch the CPU to full speed (40MHz). Some software may behave incorrectly in this mode, while other software will work very well, and run many times faster than on a C64.
- Type `POKE0,64` in C64 mode to switch the CPU to 1MHz.
- Type `SYS58552` in C64 mode to switch to C65 mode.
- Type `GO64` in C65 mode and confirm, by pressing `Y`, to switch to C64 mode, just like on a C128.

- The C65 ROM makes device 8 the default, so you can normally leave off the **,8** from the end of LOAD and SAVE commands.
- Pressing **SHIFT** + **RUN STOP** from either C64 or C65 mode will attempt to boot from disk.

Have fun! The MEGA65 has been lovingly crafted over many years for your enjoyment. We hope you have as much fun using it as we have had creating it!

The MEGA Museum of Electronic Games and Art welcomes your feedback, suggestions and contributions to this open-source digital heritage preservation project.

APPENDIX G

Flashing the FPGAs and CPLDs in the MEGA65

• Flashing the MAX10 FPGA on the MEGA65's Mainboard with INTEL QUARTUS

The MEGA65 is an open-source and open-hardware computer. This means you are free, not only to write programs that run on the MEGA65 as a finished computer, but you can use the re-programmable chips in the MEGA65 to turn it into all sorts of other things!

These re-programmable chips are called Field Programmable Gate Arrays (FPGAs) or Complex Programmable Logic Devices (CPLDs), and can implement a wide variety of circuits. They are normally programmed using a programming language like VHDL or Verilog. These are languages that are not commonly encountered by most people. They are also quite different in some ways to "normal" programming languages, and it can take a while to understand how they work, but with some effort and perseverance, many people will be able to do exiting things with them.

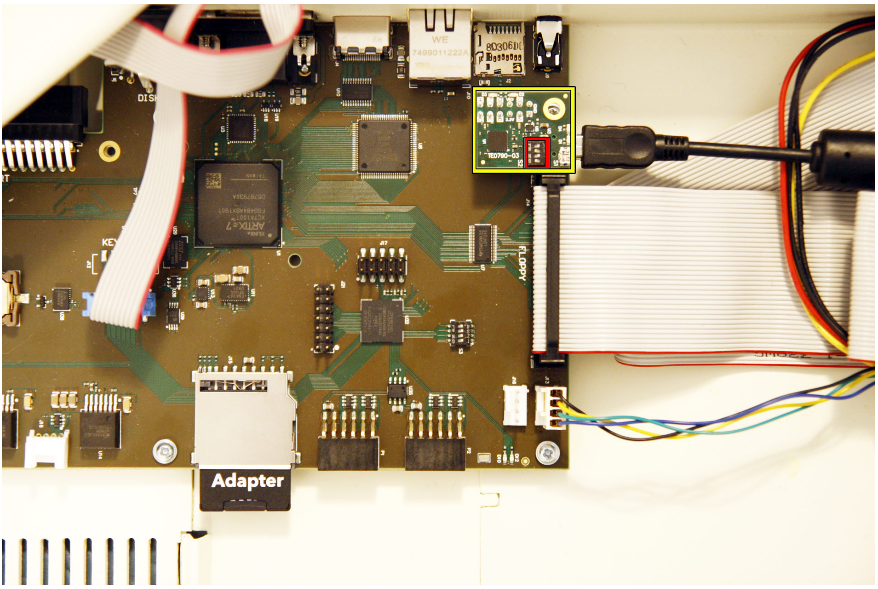
Be prepared to install many gigabytes of software on a Linux or Windows PC, before you will be able to write programs for the FPGAs and CPLDs in the MEGA65. Also, "compiling" complex designs can take up to several hours, depending on the speed and memory capacity of your computer! We recommend a computer with at least 8GB RAM, and preferably 16GB if you want to write programs for FPGAs and CPLDs. If on the other hand all you want to do is load programs onto your MEGA65's FPGAs and CPLDs that other people have written, then most computers running a recent version of Windows or Linux should be able to cope.

WARNING

Before we go any further, we do have to provide a warning about reprogramming the FPGAs and CPLDs in the MEGA65. Re-programming the MEGA65 FPGA can potentially cause damage, or leave your MEGA65 in an unresponsive state from which it is very difficult to recover, i.e., "bricked". Therefore if you choose to open your MEGA65 and reprogram any of the FPGAs it contains, it is no longer possible to guarantee its correct operation. Therefore in this case we can not reasonably honour the warranty of the device as a computer. You have been warned!

FLASHING THE ARTIX 100T MAIN FPGA WITH XILINX VIVADO

If you choose to proceed, you will need a TE0790-03 JTAG programming module, a functioning installation of Xilinx's Vivado software. This can be done on either Windows or Linux, but in both cases you will need to install any necessary USB drivers. It is also necessary to have dip-switches 1 and 3 in the ON position and dip-switches 2 and 4 in the OFF position on the TE-0790. With your MEGA65 disconnected from the power, the TE-0790 must be installed on the JB1 connector, which is located between the floppy data cable and the audio jack. The gold-plated hole of the TE-0790 must line up with the screw hole below. The mini-USB cable will then connect on the side towards the 3.5" floppy drive. The following image shows the correct position: The TE0790 is surrounded by the yellow box, and the dip-switches by the red box. Dip-switch 1 is the one nearest the floppy data cable.



Connect your non-8-bit computer to the FPGA programming device using a mini-USB cable. Open VIVADO, which can be downloaded from the internets.

To access the Hardware Manager, open a project in VIVADO or create an empty one, if you do not have any projects yet.

In the left column, select "Open Hardware Manager" at the very bottom.

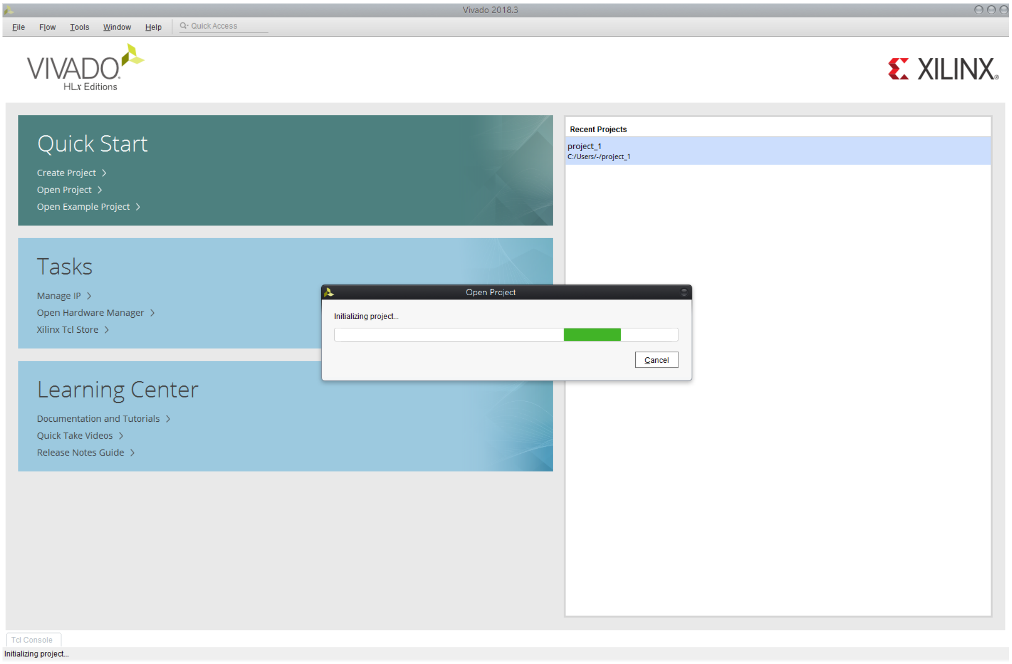
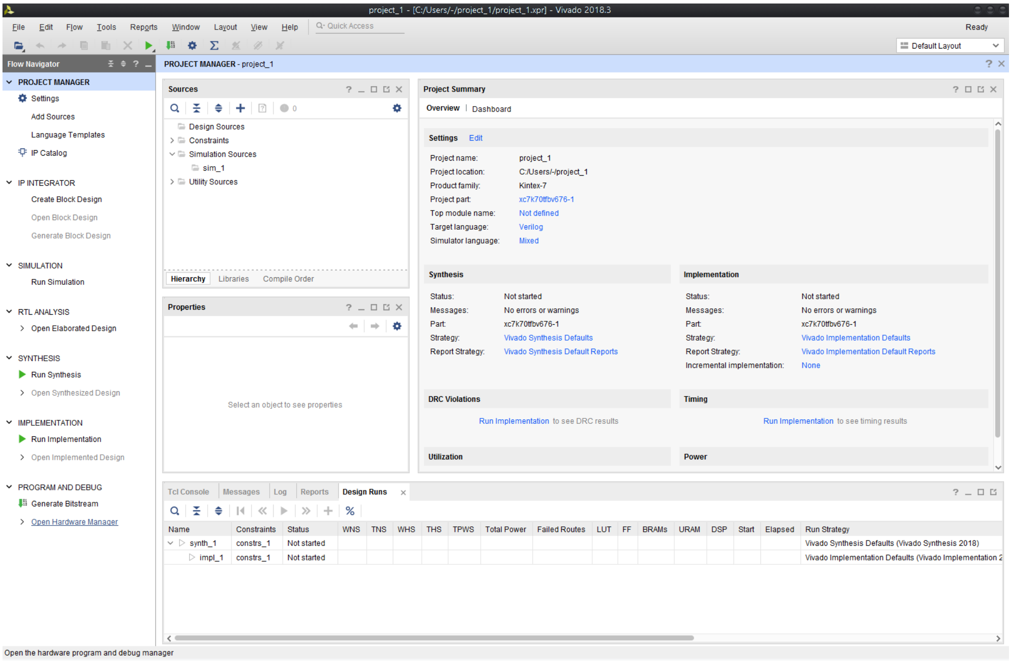


Figure G.1: Step 1: Open a project in VIVADO



Settings Edit

Project name:	project_1
Project location:	C:/Users/-/project_1
Product family:	Kintex-7
Project part:	xc7k70tfbv676-1
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7k70tfbv676-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7k70tfbv676-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

DRC Violations

Run Implementation to see DRC results

Timing

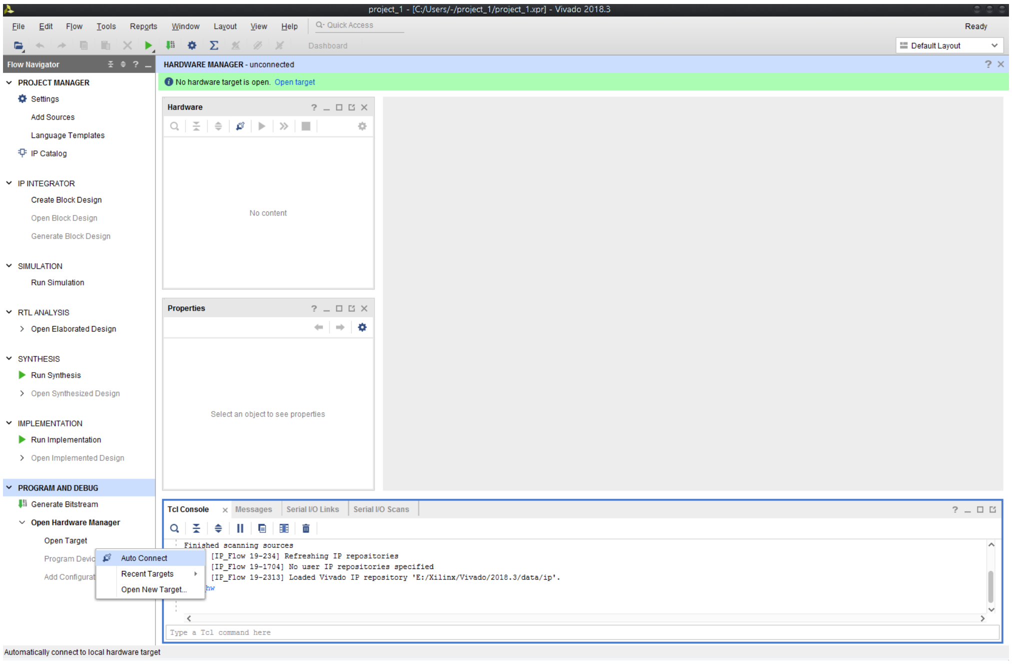
Run Implementation to see timing results

Utilization

Power

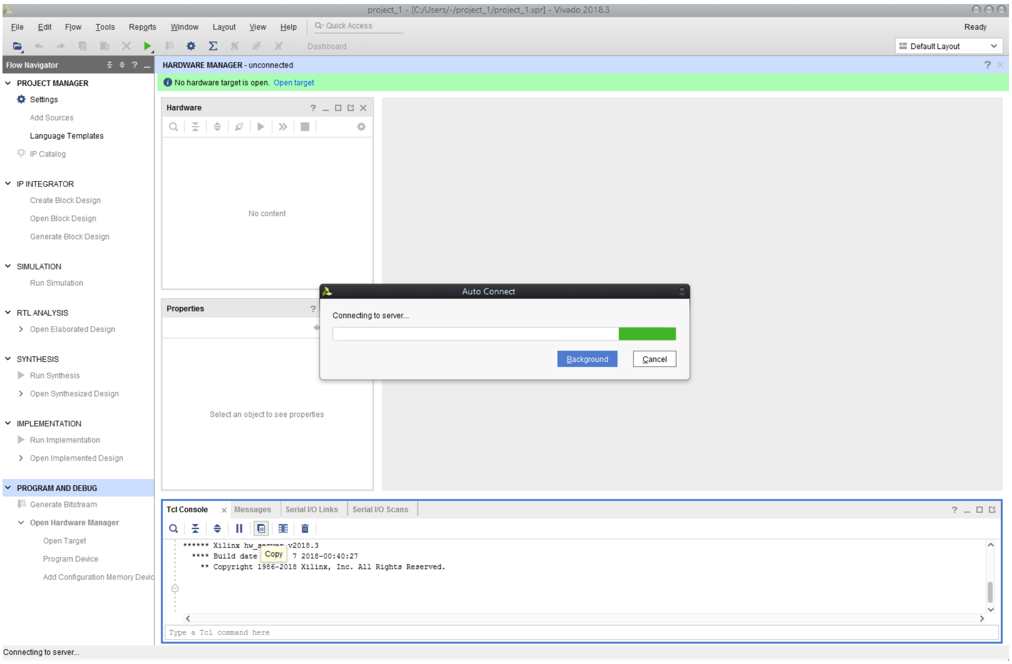
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2018)
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2

Figure G.2: Step 2: Open Hardware Manager



```
Finished scanning sources
 [IP_Flow 19-234] Refreshing IP repositories
 [IP_Flow 19-1704] No user IP repositories specified
 [IP_Flow 19-2313] Loaded Vivado IP repository 'E:/Xilinx/Vivado/2018.3/data/ip'.
hw
```

Figure G.3: Step 3: Connect to FPGA



No hardware target is open. Open target

No content

Select an object to see properties

Connecting to server...

```
****** Xilinx hw_server v2018.3
  **** Build date     7 2018-00:40:27
    ** Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
```

Type a Tcl command here

Connecting to server...

Figure G.4: Step 4: Wait a moment

Under "Hardware Manager", choose "Open Target", then "Auto Connect".

Wait a moment, "Connecting to server..." should automatically close without dropping an error to the console.

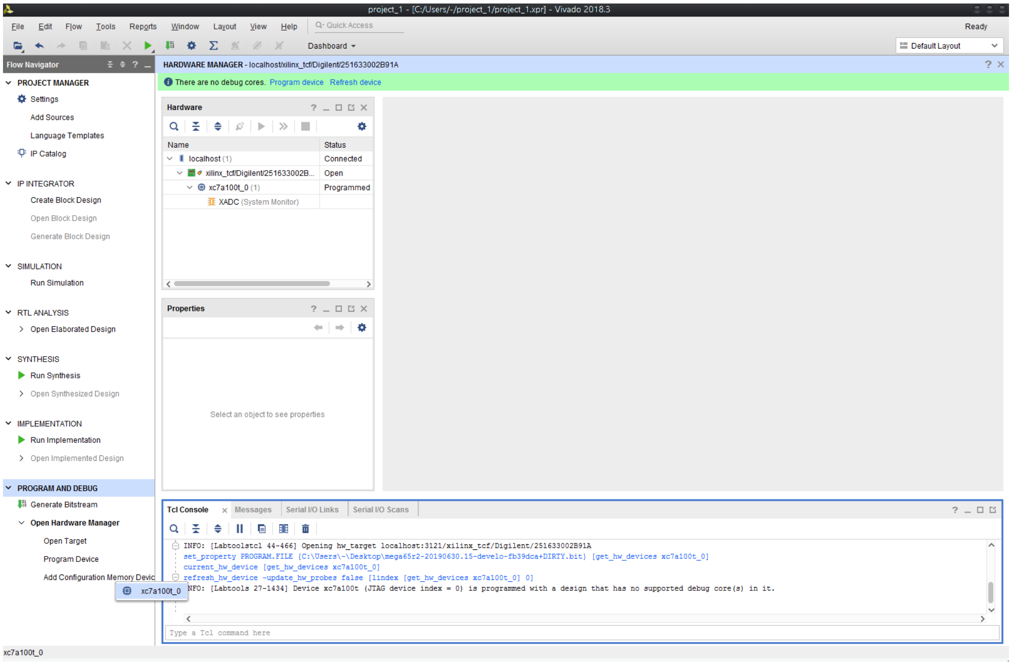
Under "Hardware Manager", choose "Add Configuration Memory Device", then "xc7a100t_0".

In the newly opened dialogue, type "S25fl256s" (without quotes), then select "s25fl256sxxxxxxx0-spi-x1_x2_x4" (the upper one) and click "OK".

In the next dialogue, choose your local Configuration file, namely a bitstream with file suffix ".mcs". Leave all other parameters as they are (see G.7).

Patiently wait for the programming to finish. This can take several minutes as the Vivado software erases and then reprograms the flash memory that is used to initialise the FPGA on power-up.

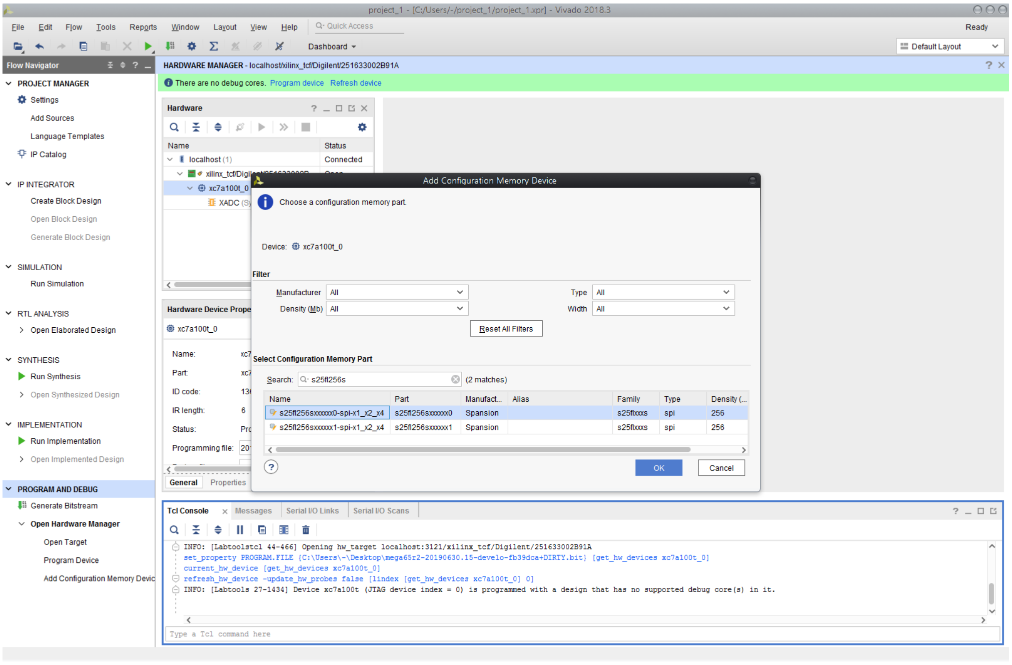
If your screen looks like G.9, your new bistream has been successfully flashed into the Artix 100T FPGA!



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/251633002B...	Open
xc7a100t_0 (1)	Programmed
XADC (System Monitor)	

```
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/251633002B91A
set_property PROGRAM.FILE {C:\Users\-\Desktop\mega65r2-20190630.15-develo-fb39dca+DIRTY.bit} [get_hw_devices xc7a100t_0]
current_hw_device [get_hw_devices xc7a100t_0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices xc7a100t_0] 0]
INFO: [Labtools 27-1434] Device xc7a100t (JTAG device index = 0) is programmed with a design that has no supported debug core(s) in it.
```

Figure G.5: Step 5: Add Configuration Memory Device


project_1 - [C:/Users/-/project_1/project_1.xpr] - Vivado 2018.3
HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/251633002B91A
There are no debug cores. Program device Refresh device
Add Configuration Memory Device
OK
Cancel

Choose a configuration memory part.

Device: xc7a100t_0

Filter

Manufacturer	All	Type	All
Density (Mb)	All	Width	All

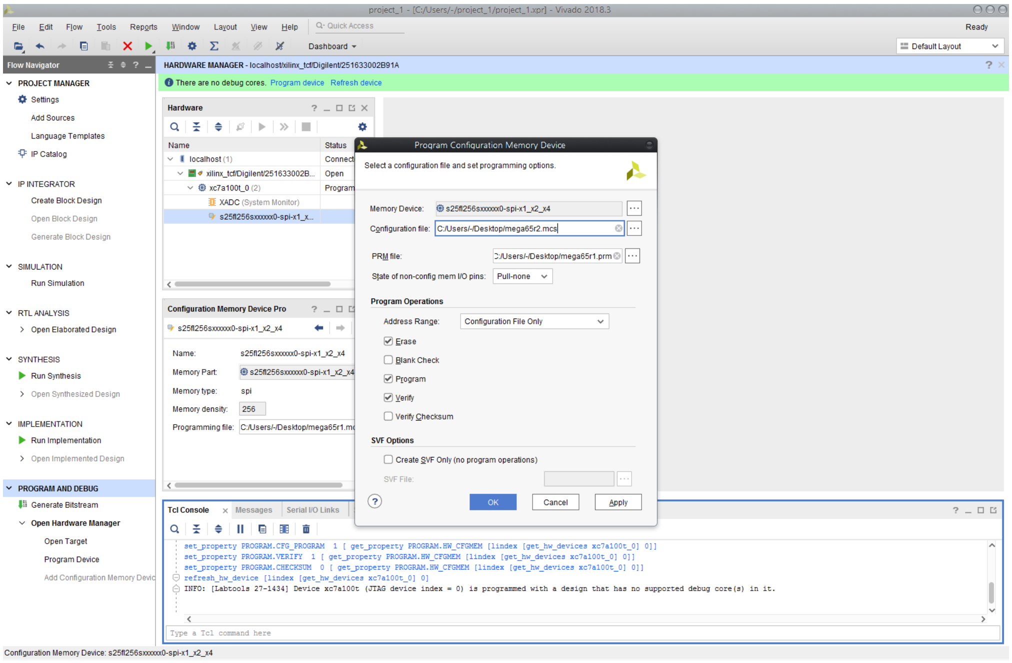
Select Configuration Memory Part

Search: s25fl256s (2 matches)

Name	Part	Manufact...	Alias	Family	Type	Density (...
s25fl256sxxxxxx0-spi-x1_x2_x4	s25fl256sxxxxxx0	Spansion		s25flxxxs	spi	256
s25fl256sxxxxxx1-spi-x1_x2_x4	s25fl256sxxxxxx1	Spansion		s25flxxxs	spi	256

```
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/251633002B91A
set_property PROGRAM.FILE {C:\Users\-\Desktop\mega65r2-20190630.15-develo-fb39dca+DIRTY.bit} [get_hw_devices xc7a100t_0]
current_hw_device [get_hw_devices xc7a100t_0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices xc7a100t_0] 0]
INFO: [Labtools 27-1434] Device xc7a100t (JTAG device index = 0) is programmed with a design that has no supported debug core(s) in it.
```

Figure G.6: Step 6: Select Memory Part


project_1 - [C:/Users/-/project_1/project_1.xpr] - Vivado 2018.3
File Edit Flow Tools Reports Window Layout View Help
Dashboard
Default Layout
Flow Navigator
HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/251633002B91A
Hardware
Configuration Memory Device Pro
Program Configuration Memory Device
OK
Cancel
Apply
Tcl Console
Messages
Serial I/O Links

There are no debug cores. Program device Refresh device

Select a configuration file and set programming options.

Memory Device: s25fl256sxxxxxx0-spi-x1_x2_x4
Configuration file: C:/Users/-/Desktop/mega65r2.mcs
PRM file: C:/Users/-/Desktop/mega65r1.prm
State of non-config mem I/O pins: Pull-none

Program Operations
Address Range: Configuration File Only
[✓] Erase
[ ] Blank Check
[✓] Program
[✓] Verify
[ ] Verify Checksum

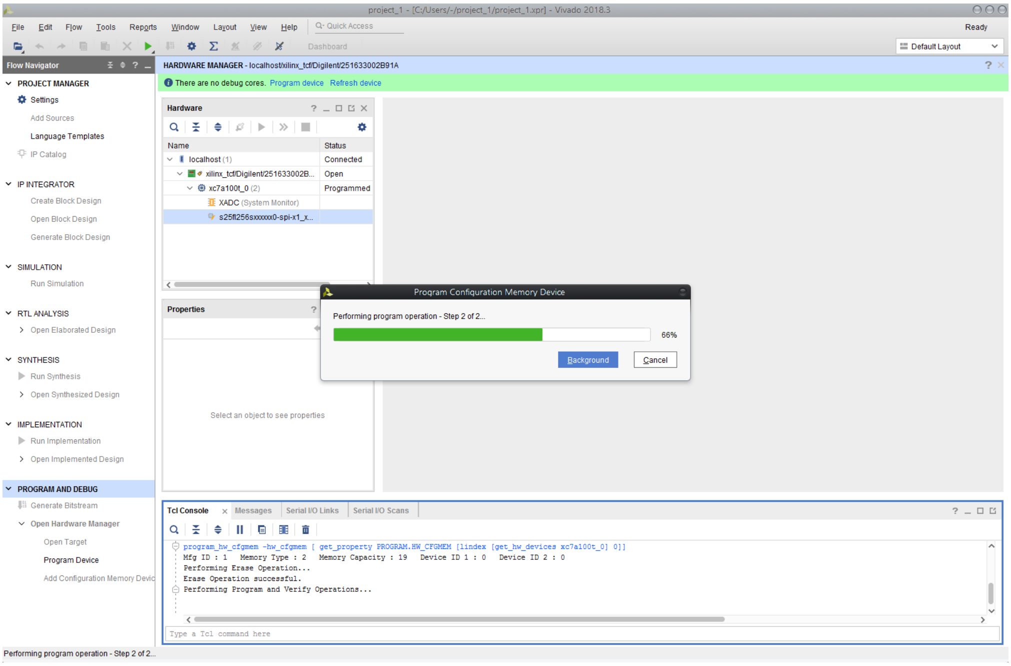
SVF Options
[ ] Create SVF Only (no program operations)
SVF File:

Name: s25fl256sxxxxxx0-spi-x1_x2_x4
Memory Part: s25fl256sxxxxxx0-spi-x1_x2_x4
Memory type: spi
Memory density: 256
Programming file: C:/Users/-/Desktop/mega65r1.mc

```
set_property PROGRAM.CFG_PROGRAM  1 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices xc7a100t_0] 0]]
set_property PROGRAM.VERIFY  1 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices xc7a100t_0] 0]]
set_property PROGRAM.CHECKSUM  0 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices xc7a100t_0] 0]]
refresh_hw_device [lindex [get_hw_devices xc7a100t_0] 0]
INFO: [Labtools 27-1434] Device xc7a100t (JTAG device index = 0) is programmed with a design that has no supported debug core(s) in it.
```

Configuration Memory Device: s25fl256sxxxxxx0-spi-x1_x2_x4

Figure G.7: Step 7: Set programming options



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/251633002B...	Open
xc7a100t_0 (2)	Programmed
XADC (System Monitor)	
s25fl256sxxxxxx0-spi-x1_x...	

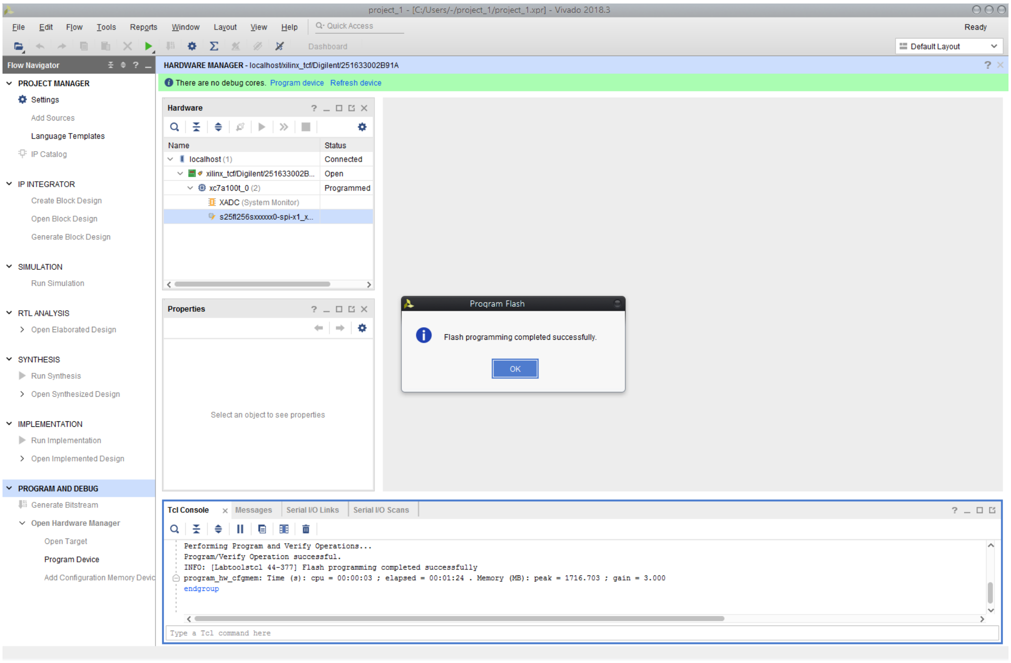
Performing program operation - Step 2 of 2...

66%

```
program_hw_cfgmem -hw_cfgmem [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices xc7a100t_0] 0]]
Mfg ID : 1   Memory Type : 2   Memory Capacity : 19   Device ID 1 : 0   Device ID 2 : 0
Performing Erase Operation...
Erase Operation successful.
Performing Program and Verify Operations...
```

Performing program operation - Step 2 of 2...

Figure G.8: Step 8: Programming in progress



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/251633002B...	Open
xc7a100t_0 (2)	Programmed
XADC (System Monitor)	
s25fl256sxxxxxx0-spi-x1_x...	

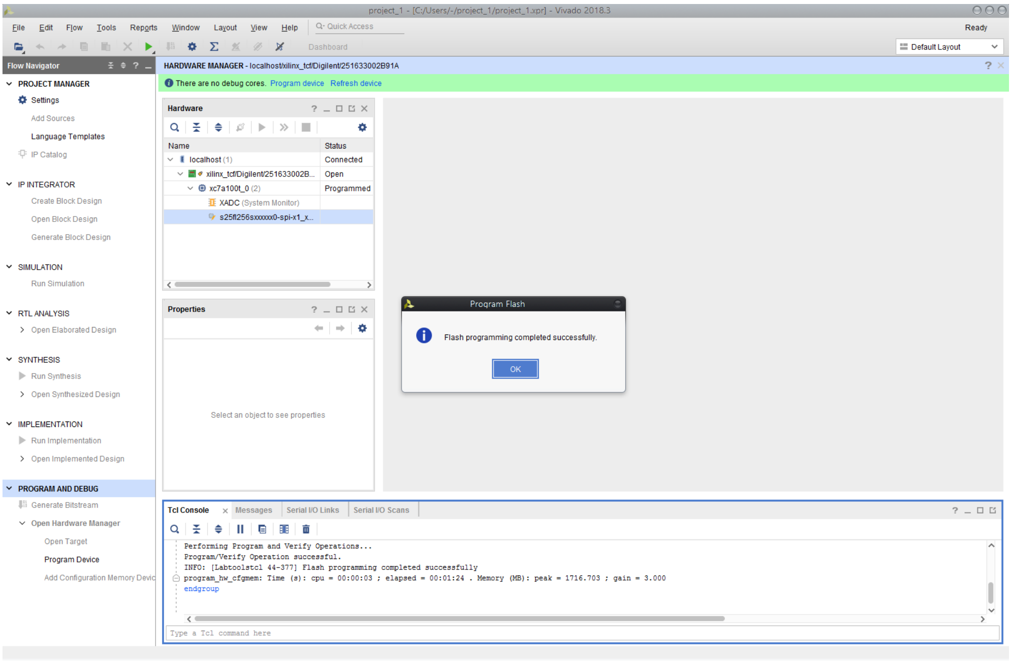
Program Flash

Flash programming completed successfully.

OK

```
Performing Program and Verify Operations...
Program/Verify Operation successful.
INFO: [Labtoolstcl 44-377] Flash programming completed successfully
program_hw_cfgmem: Time (s): cpu = 00:00:03 ; elapsed = 00:01:24 . Memory (MB): peak = 1716.703 ; gain = 3.000
endgroup
```

Figure G.9: Step 9: Programming successful



There are no debug cores. Program device Refresh device

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/251633002B...	Open
xc7a100t_0 (2)	Programmed
XADC (System Monitor)	
s25fl256sxxxxxx0-spi-x1_x...	

Flash programming completed successfully.

```
Performing Program and Verify Operations...
Program/Verify Operation successful.
INFO: [Labtoolstcl 44-377] Flash programming completed successfully
program_hw_cfgmem: Time (s): cpu = 00:00:03 ; elapsed = 00:01:24 . Memory (MB): peak = 1716.703 ; gain = 3.000
endgroup
```

Figure G.10: Step 10: Reflashing the FPGA

If you want to repeat the process, you might find the "Add Configuration Memory Device" option in step 5 greyed out. Instead, select "s25fl256sxxxxxxx0-spi-x1_x2_x4" in the "Hardware" window, press right mouse button and select "Program Configuration Memory Device" to flash.

FLASHING THE CPLD IN THE MEGA65'S KEYBOARD WITH LATTICE DIAMOND

FLASHING THE MAX10 FPGA ON THE MEGA65'S MAINBOARD WITH INTEL QUARTUS

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] X. S. me, “”vic-ii for beginners: Screen modes, cheaper by the dozen,” XXX Set me. [Online]. Available: http://dustlayer.com/vic-ii/2013/4/26/vic-ii-for-beginners-screen-modes-cheaper-by-the-dozen

INDEX

PART II

ELEMENT CATALOGUE

GRAPHIC SYMBOLS FONT

Graphic chars and MEGA logo using the **graphicsymbol** macro:

Graphic chars using the **symbolfont** font definition:

The MEGA logo in default black using the **megasymbol** macro:

for tables and symbol usage.

The MEGA logo using a passed in colour:

Special multiline keys: RUN STOP CLR HOME NO SCRL HELP INST DEL SHIFT LOCK ESC ALT

KEYBOARD KEYS

MEGA key looks like this.

NORMAL SHIFT

BIG SHIFT

Text to the left RUN STOP and text to the right.

SHIFT CTRL 9 RETURN

* ← ↑ → ↓

SCREEN OUTPUT

```
    10 INPUT A$
    20 PRINT "YOU TYPED: ";A$
    30 PRINT
    40 GOTO 10
    RUN
    ? MEGA 65
    YOU TYPED: MEGA 65
```

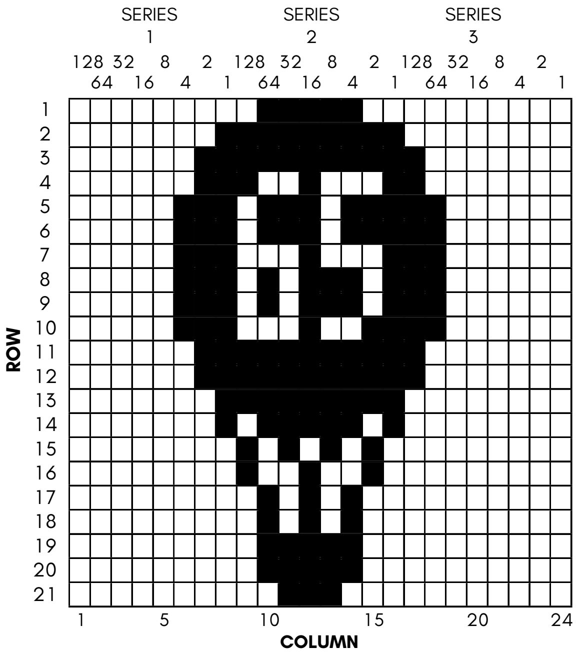
```
10 OPEN 1,8,0,"$0:*,P,R
20 : IF DS THEN PRINT DS$: GOTO 100
30 GET#1,X$,X$
40 DO
50 : GET#1,X$,X$: IF ST THEN EXIT
60 : GET#1,BL$,BH$
70 : LINE INPUT#1, F$
80 : PRINT LEFT$(F$,18)
90 LOOP
100 CLOSE 1

RUN
```

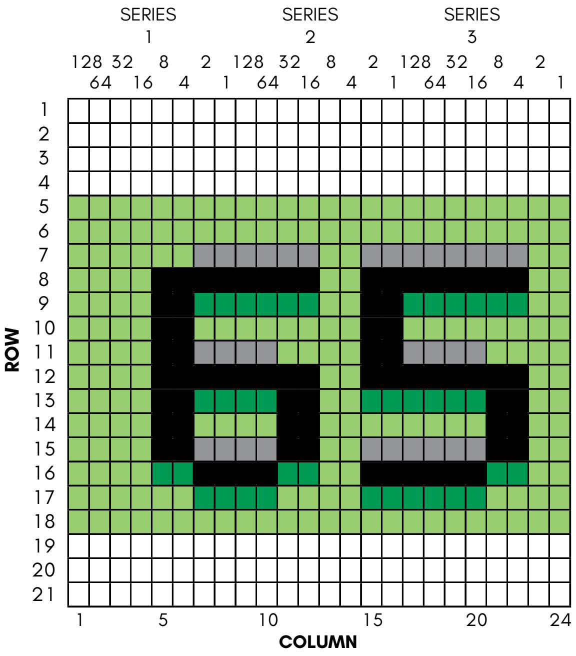
Use the "screentext" macro to perform inline screen text: `?SYNTAX ERROR`

SPRITE GRIDS

Balloon Sprite Demo



Multicolor Sprite


SERIES 1
SERIES 2
SERIES 3
128 32 8 2 128 32 8 2 128 32 8 2
64 16 4 1 64 16 4 1 64 16 4 1
ROW
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
1
5
10
15
20
24
COLUMN