C256 – Developer Introduction Notes
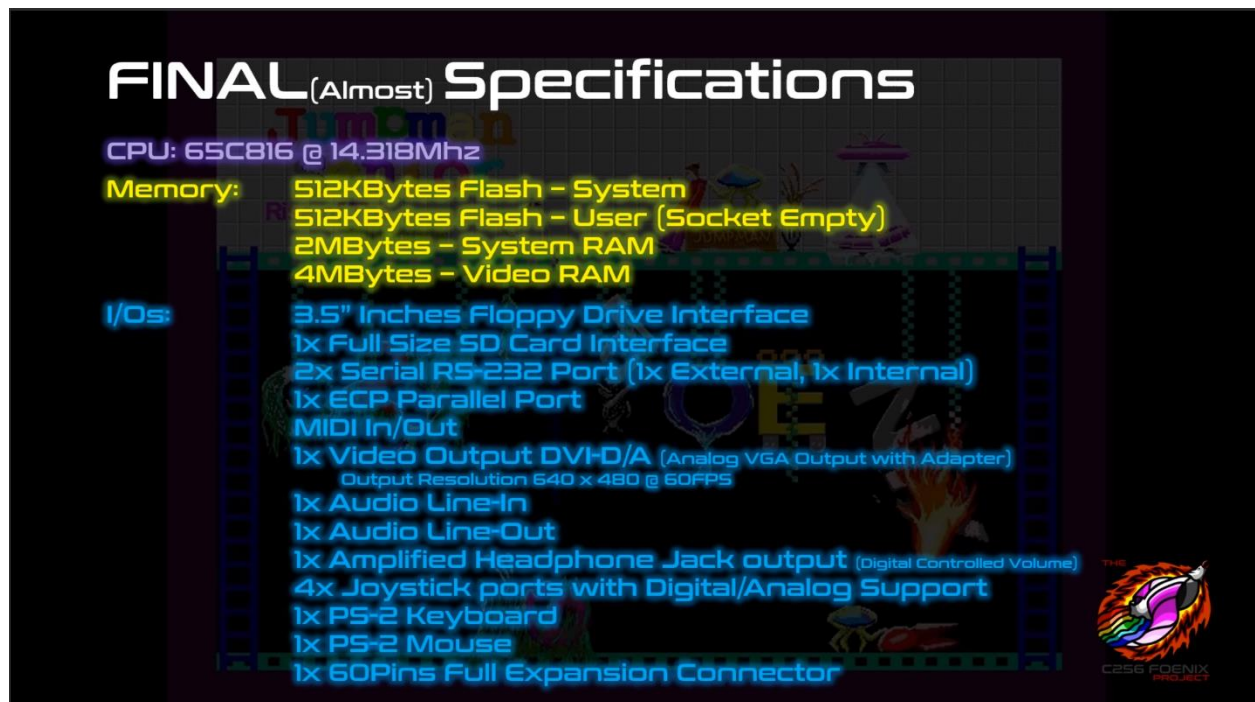
Introduction to the C256 System

Hardware

The C256 system uses a 65C816 micro-processor.

System clock is 14MHz.

Power supply required is +12V 1A with a 2.5mm plug.

Keyboard requires a PS/2 connector.

Contributing to the IDE Development

Git Repositories

Foenix IDE: https://github.com/Trinity-11/FoenixIDE.git

C256 Kernel: https://github.com/Trinity-11/Kernel

Tools

To modify the Foenix IDE, you will need Visual Studio 2017 Community edition for C#.

Using the Foenix IDE

The Integrated Development Environment consists of a 65816 emulator and a C256 Foenix memory map emulator.

Launching the Application

To start the IDE, double-click on the FoenixIDE icon or using a console, type "FoenixIDE.exe".

The command-line accepts three parameters:

-h, --hex: load the program into memory for an "Intel Hex" file format;

-r, --run: auto-run the provided binary; and

-i, --irq: disable "break on interrupts" in the CPU window.

If no parameters are specified the application will launch with defaults:

- Program is loaded from ROMS\kernel.hex; if it doesn't exist, the user is prompted to select one using the Windows File Dialog;
- Autorun is disabled; and
- Break on Interrupts is enabled.

IDE Windows

The IDE consists of three main windows. The display, the CPU debugger and the memory editor.

Understanding the C256 Foenix

The Foenix IDE currently emulates the C256 Foenix computer Revision B. Once Revision C is available, this document and the IDE will need to be updated.

Memory Map

The CPU can access 24-bit worth of addresses.
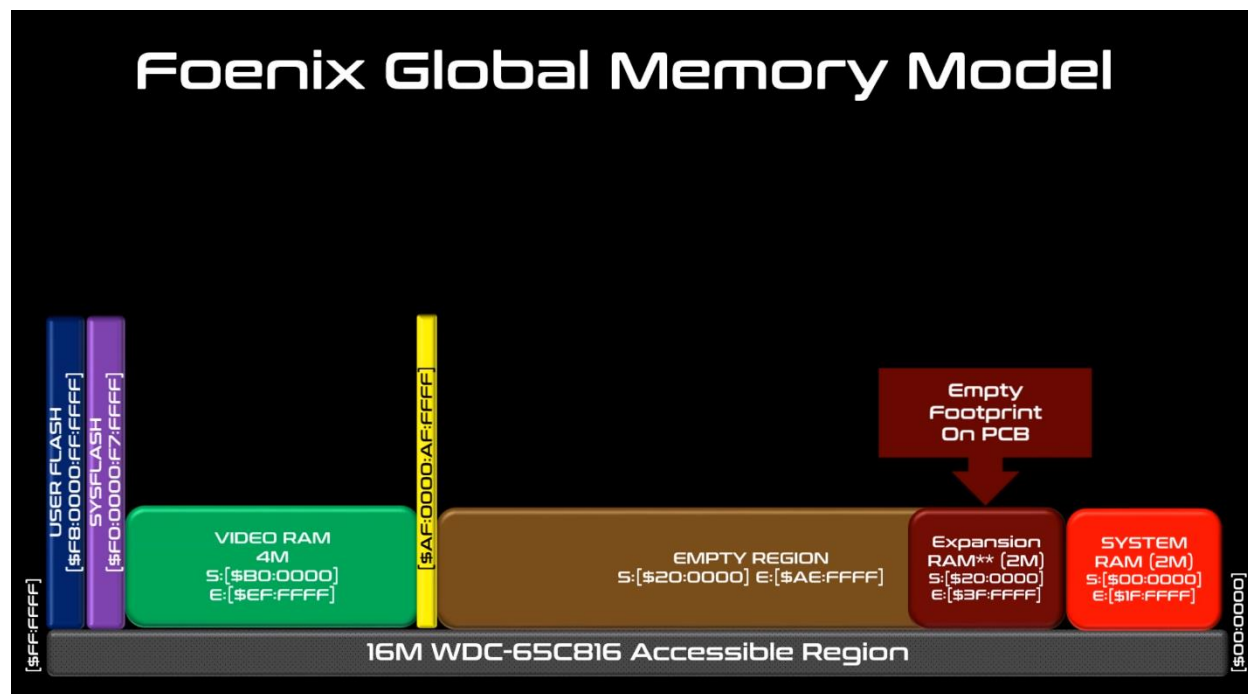
$FF:0000 - $FF:FFFF	Bank $FF	
$FE:0000 - $FE:FFFF	Bank $FE	
		16 MB Address Space
$00:0000 - $01:FFFF	Bank $01	
$00:0000 - $00:FFFF	Bank $00	

The address space is mapped as follows:

$F8:0000 - $FF:FFFF	512 KB User Flash (if populated)
$F0:0000 - $F7:FFFF	512 KB System Flash
$B0:0000 - $EF:FFFF	4 MB Video RAM
$AF:0000 - $AF:FFFF	IO Space
$40:0000 - $AE:FFFF	Extension Card
$20:0000 - $3F:FFFF	2 MB RAM (optional)
$00:0000 - $1F:FFFF	2 MB RAM

On boot, Gavin copies the first 64KB of the content of System Flash (or User Flash, if present) to Bank $00. The entire 512KB are copied to address range $18:0000 to $1F:FFFF.

IO Space is mapped to Vicky: $AF:0000 to $AF:DFFF and Beatrix: $AF:E000 to $AF:FFFF.

Gavin – Location $00:0000 to $00:FFFF
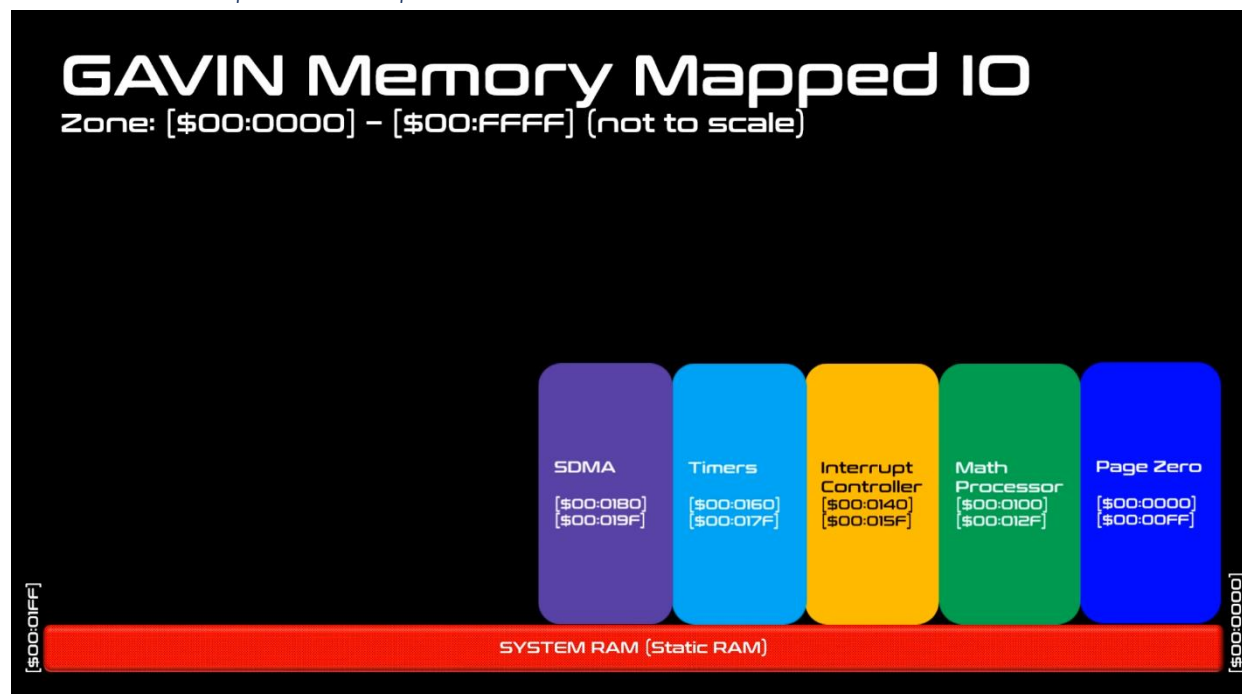
Math Co-Processor

The C256 provides a math co-processor to perform long addition, multiplications and divisions.

To perform an operation, you write the little-endian values in the appropriate address locations and the results are automatically returned in the result addresses.

Multiplications

There are two multiplier locations: $00:0100 and $00:0108. Multiplier 0 is unsigned and Multiplier 1 is signed. Each operand must be 16-bits, and the result is 32-bits.

Address	Name	Description
$00:0100	M0_OPERAND_A	16-bit unsigned value
$00:0102	M0_OPERAND_B	16-bit unsigned value
$00:0104	M0_RESULT	32-bit unsigned result of the multiplication of A and B

Address	Name	Description
$00:0108	M1_OPERAND_A	16-bit signed value
$00:010A	M1_OPERAND_B	16-bit signed value
$00:010C	M1_RESULT	32-bit signed result of the multiplication of A and B

Divisions

There are two divider locations: $00:0110 and $00:0118. Divider 0 is unsigned and Divider 1 is signed. Each operand must be 16-bits. The result and remainder are 16-bits also.

Address	Name	Description
$00:0110	D0_OPERAND_A	16-bit unsigned value for the dividend
$00:0112	D0_OPERAND_B	16-bit unsigned value for the divisor
$00:0114	D0_RESULT	16-bit unsigned result for the quotient
$00:0116	D0_REMAINDER	16-bit unsigned result for the remainder

Address	Name	Description
$00:0118	D1_OPERAND_A	16-bit signed value for the dividend
$00:011A	D1_OPERAND_B	16-bit signed value for the divisor
$00:011C	D1_RESULT	16-bit signed result for the quotient
$00:011E	D1_ REMAINDER	16-bit signed result for the remainder

Long Signed Additions

There is one long signed adder located at $00:0120. Both operands must be 32-bit signed integers. The result is also 32-bit signed.

| Address	Name	Description
$00:0120	ADDER32_A	32-bit signed value
$00:0124	ADDER32_B	32-bit signed value
$00:0128	ADDER32_R	32-bit signed result of the addition of A and B

Interrupt Controller

Address	Name	Description
$00:0140	INT_PENDING_REG0	Pending Interrupts Register 0
$00:0141	INT_PENDING_REG1	Pending Interrupts Register 1
$00:0142	INT_PENDING_REG2	Pending Interrupts Register 2
$00:0144	INT_POL_REG0	Polarity Set Interrupts Register 0
$00:0145	INT_POL_REG1	Polarity Set Interrupts Register 1
$00:0146	INT_POL_REG2	Polarity Set Interrupts Register 2
$00:0148	INT_EDGE_REG0	Edge Interrupts Register 0
$00:0149	INT_EDGE_REG1	Edge Interrupts Register 1
$00:014A	INT_EDGE_REG2	Edge Interrupts Register 2
$00:014C	INT_MASK_REG0	Interrupt Masks Register 0
$00:014D	INT_MASK_REG1	Interrupt Masks Register 1
$00:014E	INT_MASK_REG2	Interrupt Masks Register 2

Interrupt Register 0 ($00:0140, $00:0144, $00:0148, $00:014C)							
7	6	5	4	3	2	1	0
Mouse	Floppy Disk	RTC	Timer 2	Timer 1	Timer 0	SOL	SOF

Description

SOF	Start of Frame – 60 Frames per Second (FPS)
SOL	Start of Line – 60 FPS offset
RTC	Clock Alarm, etc..., See the chip spec for more detail for the registers: BQ4802

C256 Foenix VICKY Display Arrangement
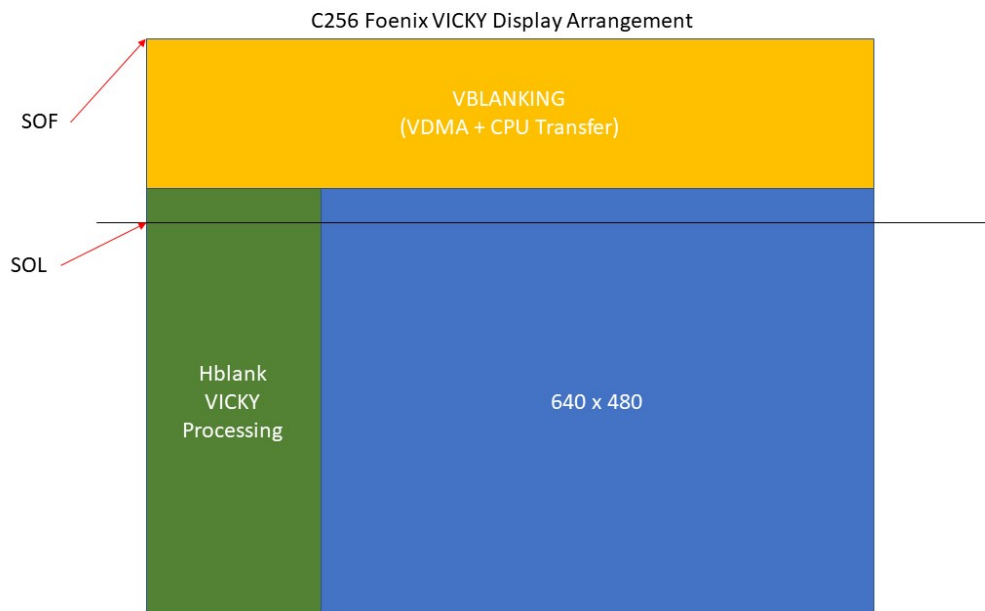
Interrupt Register 1 ($00:0141, $00:0145, $00:0149, $00:014D)							
7	6	5	4	3	2	1	0
SDCARD	LPT1	MPU-401	COM1	COM2	Tile Coll	Sprite Coll	Keyboard

Interrupt Register 2 ($00:0142, $00:0146, $00:014A, $00:014E)							
7	6	5	4	3	2	1	0
Always 1	Expansion	DAC	Video DMA	Gavin DMA	Beatrix	OPL2 Left	OPL2 Right

Vicky – Location $AF:0000 to $AF:DFFF

VICKY Memory Mapped IO
Zone: [$AF:0000] – [$AF:DFFF] (not to scale)

Vicky Memory

$AF:0000 - $AF:00FF (Internal Memory) Vicky General Registers

BORDER_COLOR_B = $AF:0005 - when in text mode, this is the border color shown.
       BORDER_COLOR_G = $AF:0006
       BORDER_COLOR_R = $AF:0007

When in Graphic Mode, if a pixel is "0" then the Background pixel is chosen
       BACKGROUND_COLOR_B   = $AF:000D
       BACKGROUND_COLOR_G   = $AF:000E
       BACKGROUND_COLOR_R   = $AF:000F

$AF:0100 - $AF:013F (Internal Memory) Vicky Tiles Registers
$AF:0140 - $AF:014F (Internal Memory) Vicky Bitmap Registers
$AF:0200 - $AF:03FF (Internal Memory) Vicky Sprites
$AF:0400 - $AF:04FF (Internal Memory) Vicky VDMA
$AF:0800 - $AF:080F Real-time clock (RTC)

Video Direct Memory Access

Memory Address Range reserved: DMA Controller $AF0400 - $AF04FF

```
VDMA_CONTROL_REG        = $AF0400
```

VDMA Control Register ($AF:0400)							
7	6	5	4	3	2	1	0
Start Tfr	-	-	-	IRQ Enable	Fill	2D	Enable

Description

VDMA_CTRL_Enable	Enable the VDMA Transfer
VDMA_CTRL_1D_2D	0 - 1D (Linear) Transfer , 1 - 2D (Block) Transfer
VDMA_CTRL_TRF_Fill	0 - Transfer Src -> Dst, 1 - Fill Destination with "Byte2Write"
VDMA_CTRL_Int_Enable	Set to 1 to Enable the Generation of Interrupt when the Transfer is over
VDMA_CTRL_Start_TRF	Set to 1 To Begin Process, Need to Cleared before, you can start another

```
VDMA_BYTE_2_WRITE   = $AF0401 ; Write Only - Byte to Write in the Fill Function
VDMA_STATUS_REG     = $AF0401 ; Read only
```

VDMA Status Register ($AF:0401) – Read-only							
7	6	5	4	3	2	1	0
Tfr In Progress	-	-	-	-	Invalid Src Addr	Invalid Dest Addr	Size Error

When Transfer is in Progress, the CPU will not be able to access Video Memory.

```
VDMA_SRC_ADDY_L     = $AF0402 ; Pointer to the Source of the Data to be transferred
VDMA_SRC_ADDY_M     = $AF0403 ; This needs to be within Vicky's Range ($00:0000 -
$3F:0000)
VDMA_SRC_ADDY_H     = $AF0404


VDMA_DST_ADDY_L     = $AF0405 ; Destination Pointer within Vicky's video memory Range
VDMA_DST_ADDY_M     = $AF0406 ; ($00:0000 - $3F:0000)
VDMA_DST_ADDY_H     = $AF0407
```

In 1D Transfer Mode, specify how many bytes to transfer. Maximum value is $40:0000 or 4 megabytes.
```
VDMA_SIZE_L         = $AF0408
VDMA_SIZE_M         = $AF0409
VDMA_SIZE_H         = $AF040A
VDMA_IGNORED        = $AF040B
```

In 2D Transfer Mode, specify the width and height to transfer. Maximum is 65,536 in each direction.
```
VDMA_X_SIZE_L       = $AF0408 ; Maximum Value: 65535
VDMA_X_SIZE_H       = $AF0409
VDMA_Y_SIZE_L       = $AF040A ; Maximum Value: 65535
VDMA_Y_SIZE_H       = $AF040B


VDMA_SRC_STRIDE_L   = $AF040C ; Always use an Even Number ( The Engine uses Even Ver
of that value)
VDMA_SRC_STRIDE_H   = $AF040D ;
VDMA_DST_STRIDE_L   = $AF040E ; Always use an Even Number ( The Engine uses Even Ver
of that value)
VDMA_DST_STRIDE_H   = $AF040F ;
```

$AF:1000 - $AF:13FF - SUPER IO Devices --- (TOTAL USAGE) ---
// Super IO Details:
// $AF:1060 - $AF:1064 - LOGIC DEVICE 7 - KEYBOARD
// $AF:1100 - $AF:117F - LOGIC DEVICE A - PME (Runtime Registers)
// $AF:1200 - $AF:1200 - LOGIC DEVICE 9 - GAME PORT
// $AF:12F8 - $AF:12FF - LOGIC DEVICE 5 - SERIAL 2
// $AF:1330 - $AF:1331 - LOGIC DEVICE B - MPU-401
// $AF:1378 - $AF:137F - LOGIC DEVICE 3 - PARALLEL PORT
// $AF:13F0 - $AF:13F7 - LOGIC DEVICE 0 - FLOPPY CONTROLLER
// $AF:13F8 - $AF:13FF - LOGIC DEVICE 4 - SERIAL 1

```
$AF:1F00 - $AF:1F3F (Internal Memory) Vicky Text Mode 16 Color Look-up Table
Foreground Color
$AF:1F40 - $AF:1F7F (Internal Memory) Vicky Text Mode 16 Color Look-up Table
Background Color
$AF:2000 - $AF:23FF (Internal Memory) Graphic Mode LUT0
$AF:2400 - $AF:27FF (Internal Memory) Graphics Mode LUT1
```

```
$AF:2800 - $AF:2BFF (Internal Memory) Graphics Mode LUT2
$AF:2C00 - $AF:2FFF (Internal Memory) Graphics Mode LUT3
$AF:4000 - $AF:40FF (External Memory) 256 Bytes GAMMA LUT - RED
$AF:4100 - $AF:41FF (External Memory) 256 Bytes GAMMA LUT - GREEN
$AF:4200 - $AF:42FF (External Memory) 256 Bytes GAMMA LUT – BLUE
```

FONT_MEMORY_BANK0   = $AF:8000 - $AF:8FFF

FONT_MEMORY_BANK1   = $AF:9000 - $AF:9FFF

Screen Page 0 – Location $AF:A000

Screen Page 0 memory is used to store text characters for display.

One page of text is 128 columns by 64 rows. This adds up to 8 KB of memory of text. C256 does not display the entire buffer on the screen. Typically, we render 72 characters per row, with 56 rows.

```
This uses 576 x 448 of the available 640 x 480 resolution.  The border size can be
modified or turned off completely.
```

```
The display process reads Screen Page 0 and for each character, displays it's
character set bitmap.
```

Screen Page 1 – Location $AF:C000

An additional page of 128 x 64 is used to store the colors. Each byte is split into foreground (4bits) and background (4 bits). The high nibble (bits 7..4) are the foreground and the low nibble (bits 3..0) are the background.

The colors used (the 4 bits) are used to lookup RBG values in two lookup tables (LUT).

The foreground (FG) LUT is located at $AF:1F40 for 64 bytes – only 16 x 3 = 48 bytes are used. The extra byte may be used for alpha (transparency) later on.

The background (BG) LUT is located at $AF:1F80 for 64 bytes – only 16 x 3 = 48 bytes are used. The extra byte may be used for alpha (transparency) later on.

The colors are assigned 8-bit blue, 8-bit green, 8-bit red, 8-bit alpha (not used) for each of those colors in Text Mode.

Example – Color Lookup

Consider the following Foreground and Background Lookup Tables

Foreground Color Lookup Table, starting at address $AF:1F40						Background Color Lookup Table, starting at address $AF:1F80					
Index	Blue	Green	Red	Alpha	Color	Index	Blue	Green	Red	Alpha	Color
0	$00	$00	$00	$FF	Black	0	$00	$00	$00	$FF	Black
1	$00	$00	$80	$FF	Maroon	1	$00	$00	$80	$FF	Maroon
2	$00	$80	$00	$FF	Green	2	$00	$80	$00	$FF	Green
3	$80	$00	$00	$FF	Navy	3	$80	$00	$00	$FF	Navy
4	$00	$80	$80	$FF	Olive	4	$00	$20	$20	$FF	??
5	$80	$80	$00	$FF	Teal	5	$20	$20	$00	$FF	??
6	$80	$00	$80	$FF	Purple	6	$20	$00	$20	$FF	??
7	$80	$80	$80	$FF	Gray	7	$20	$20	$20	$FF	??
8	$00	$45	$FF	$FF	Orange	8	$1E	$69	$D2	$FF	
9	$13	$45	$8B	$FF	Brown	9	$13	$45	$8B	$FF	Brown
A	$00	$00	$20	$FF	Dark Red	A	$00	$00	$20	$FF	Dark Red
B	$00	$20	$00	$FF	Dark Green	B	$00	$20	$00	$FF	Dark Green
						C	$40	$00	$00	$FF	Blue

C	$20	$00	$00	$FF	Indigo
D	$20	$20	$20	$FF	Dark Gray
E	$40	$40	$40	$FF	Slate Gray
F	$FF	$FF	$FF	$FF	White

D	$10	$10	$10	$FF	Midnight Gray
E	$40	$40	$40	$FF	Slate Gray
F	$FF	$FF	$FF	$FF	White

If a character in Screen Page 1 is $ED (the default text color combination), then the foreground color index is E and the background color index is D. Looking up the index for E will make the foreground "Slate Gray" and the background "Midnight Gray". The image below shows this color combination in text.
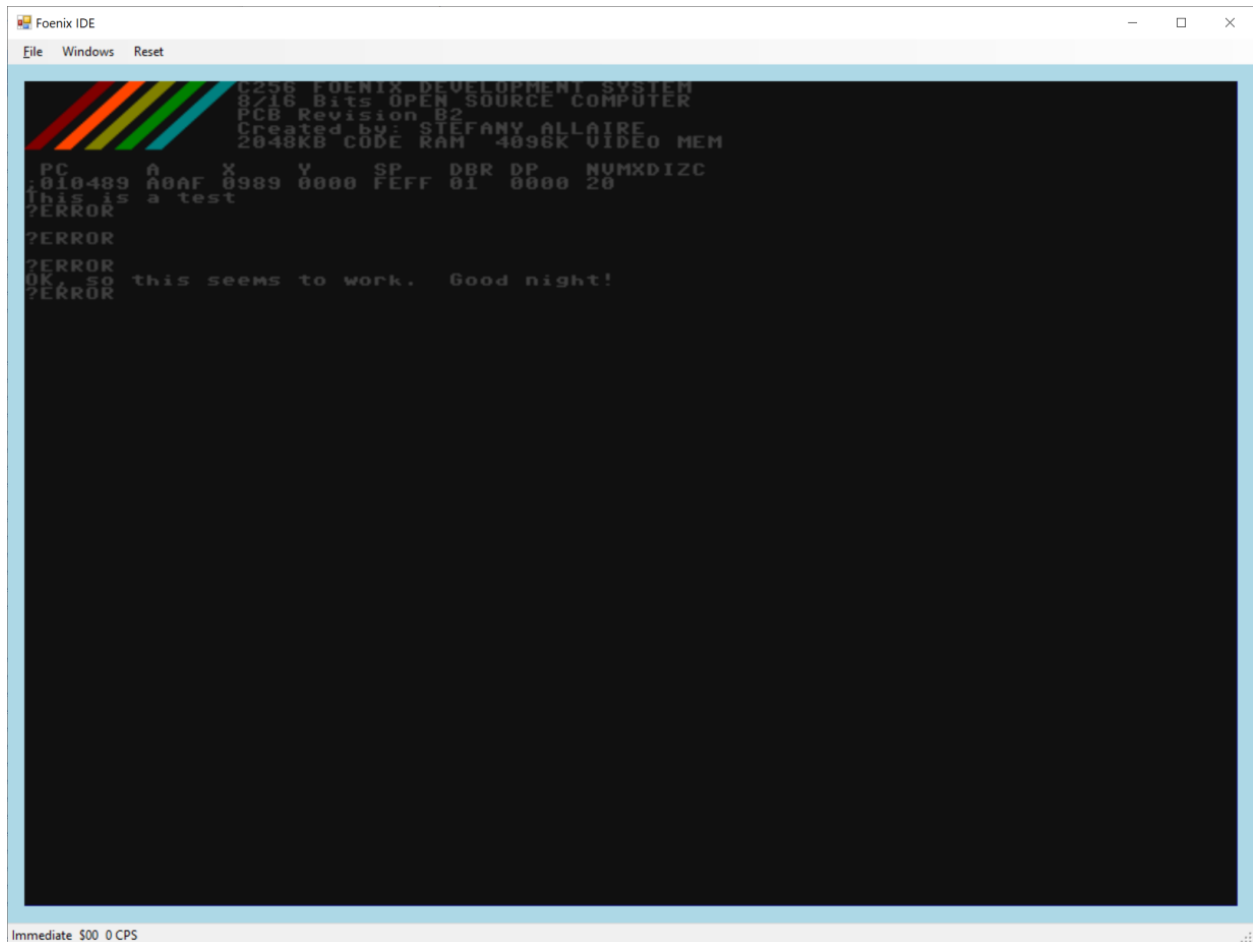
Text Gamma Lookup Table

The Gamma lookup table is used to adjust the color between different display devices (such as DVI versus VGA). Each of the red, green and blue can be corrected. Each table consists of 256 values.

```
GAMMA_B_LUT_PTR             = $AF:4000
GAMMA_G_LUT_PTR             = $AF:4100
GAMMA_R_LUT_PTR             = $AF:4200
```

Gamma can be enabled or disabled.

Master Control Register

The Master Control Register (MCR) is used to enable/disable various video mode. The MCR is located at address $AF:0000.

Master Control Register ($AF:0000)							
7	6	5	4	3	2	1	0

Disable Vid	Gamma	Sprite	Tilemap	Bitmap	Graph Mode	Text Overlay	Text Mode

MCR Bit	MCR Name	Description
0	Mstr_Ctrl_Text_Mode_En	Enable the Text Mode
1	Mstr_Ctrl_Text_Overlay	Enable the Overlay of the text mode on top of Graphic Mode (the Background Color is ignored)
2	Mstr_Ctrl_Graph_Mode_En	Enable the Graphic Mode
3	Mstr_Ctrl_Bitmap_En	Enable the Bitmap Module in Vicky
4	Mstr_Ctrl_TileMap_En	Enable the Tile Module in Vicky
5	Mstr_Ctrl_Sprite_En	Enable the Sprite Module in Vicky
6	Mstr_Ctrl_GAMMA_En	Enable the GAMMA correction - The Analog and DVI have different color value, the GAMMA is great to correct the difference. NOTE: This could also be used for fade-in and out.
7	Mstr_Ctrl_Disable_Vid	This bit disables the Scanning of the Video Memory, hence giving 100% bandwidth to the CPU to access graphic data. NOTE: In this case the Border color or the background is displayed on the screen (I can't remember) to be advised

Displaying Graphics

C256 has 4 MB of Video RAM available, starting at $B0:0000 and ending at $EF:FFFF.

The order in which images are drawn are:

- L0 - Sprites (Closest to the Viewer)
- L1 - Tile Layer 0
- L2 - Sprite Layers
- L3 - Tile Layer 1
- L4 - Sprite Layers
- L5 - Tile Layer 2
- L6 - Sprite Layers
- L7 - Tile Layer 3
- L8 - Sprite Layers
- L9 - Background (bitmap)

Bitmaps

Once the Bitmap bit is set in the MCR, Vicky will retrieve the Bitmap Control Register at address $AF:0140.

The Bitmap Control Register is shown below.

Bitmap Control Register ($AF:0140)		
7 .. 4	3 .. 1	0
Reserved	LUT	Enable

BCR Bit	BCR Name	Description
0	Enabled	Enable the bitmap
1 .. 3	LUT	Lookup Table Index 0 to 7
4 .. 7	Reserved	Reserved

The address pointer of the bitmap in the Video RAM is located at addresses $AF:0141 to $AF:0143. The address stored must be offset by $B0:0000. As an example, if the bitmap data is stored in at address $B1:4000 in memory, the address pointer must be $01:4000.

The bitmap width is saved in the word $AF:0144 to $AF:0145.

The bitmap height is saved in the word $AF:0146 to $AF:0147.

Tiles

Once the Tile bit is set in the MCR, Vicky will retrieve the Tile Control Register at addresses $AF:0100, $AF:0108, $AF:0110 and $AF:0118 to determine if a tileset should be displayed. There can be four tilesets at any given time in the display.

The Tile Control Registers are shown below.

Tile Control Register ($AF:0100, $AF:0108, $AF:0110, $AF:0118)			
7	6 .. 4	3 .. 1	0
Tile Striding	Reserved	LUT	Enable

TCR Bit	TCR Name	Description
0	Enabled	Enable the bitmap
1 .. 3	LUT	Lookup Table Index 0 to 7
4 .. 6	Reserved	Reserved
7	Tile Striding	0: sequential, 1: 256 x 256 Tile sheet striding

Each tile has its own control register, video address pointer, X and Y stride.

	Tile 0	Tile 1	Tile 2	Tile 3
Tile Control Register	$AF:0100	$AF:0108	$AF:0110	$AF:0118
Tile Start Address	$AF:0101 .. $AF:0103	$AF:0109 .. $AF:010B	$AF:0111 .. $AF:0113	$AF:0119 .. $AF:011B
Tile Map X Stride	$AF:0104 .. $AF:0105	$AF:010C .. $AF:010E	$AF:0114 .. $AF:0115	$AF:011C .. $AF:011E
Tile Map Y Stride	$AF:0106 .. $AF:0107	$AF:010E .. $AF:010F	$AF:0116 .. $AF:0117	$AF:011E .. $AF:011F

Tile maps are stored at addresses $AF:5000, $AF:5800, $AF:6000 and $AF:6800.

Sprites

Once the Sprite bit is set in the MCR, Vicky will display sprites in the appropriate layer. There can be 32 sprites displayed for each screen refresh.

The Sprite Control Registers are shown below.

Sprite Control Register ($AF:0200 to $AF:02F8, offset by 8 bytes)			
7	6 .. 4	3 .. 1	0
Reserved	Layer	LUT	Enable

Each sprite has a Control Register, a video memory address, and X and Y locations.

Beatrix

BEATRIX Memory Mapped IO
Zone: [$AF:E000] – [$AF:FFFF] (not to scale)

| | CODEC DAC OUTPUT FIFO TBD | CODEC ADC INPUT FIFO TBD | CODEC REGISTERS [$EB20]-[$EB23] | SDCARD CONTROLLER [$EB08]-[$EB0F] | IO PORT (JOYSTICKS) [$EB00]-[$EB07] | OPL2 – BOTH (WRITE) [$E700]-[$E7FF] | OPL2 - LEFT [$E500]-[$E6FF] | OPL2 - RIGHT [$E500]-[$E5FF] | SID [$E400]-[$E4FF] |

16M WDC-65C816 Accessible Region – Page [$AF]

[$AF:FFFF] ... [$AF:E000]

```
// $AF:E400..$AF:E4FF    // SID (Still to be defined) Not Implemented yet
// $AF:E500..$AF:E5FF    // OPL2 - Left Side
// $AF:E600..$AF:E6FF    // OPL2 - Right Side
// $AF:E700..$AF:E7FF    // OPL2 - Both Side (Write Sequence Only)
// $AF:E800..$AF:E807    // Joystick + AD Channel + SD Controller
// $AF:E820..$AF:E823    // CODEC Register
// $AF:E810..$AF:E81F    //SD Card Stat
```

Joystick Ports

The C256 Foenix provides 4 joystick ports, located in front of the machine. The 9-pin connectors are compatible with the standard Atari/Commodore joysticks. Joystick ports are number from right to left, starting near the Reset button.

```
JOYSTICK0      = $AF:E800  ;(R) Joystick 0 - J7 (Next to Buzzer)
JOYSTICK1      = $AF:E801  ;(R) Joystick 1 - J8
JOYSTICK2      = $AF:E802  ;(R) Joystick 2 - J9
JOYSTICK3      = $AF:E803  ;(R) Joystick 3 - J10 (next to SD Card)
```

The joysticks do not generate interrupts. Polling ports at a regular interval, say with the SOF interrupt is recommended.

The joystick register has a value at rest of $9F. The bits are set to zero when a switch is closed.

Joystick Register ($AF:E800, $AF: E801, $AF: E802, $AF: E803)							
7	6	5	4	3	2	1	0
Button	N/A	N/A	N/A	Left	Right	Down	Up

Note: This feature is not implemented in the Foenix IDE yet.

```
DIPSWITCH     = $AF:E804  ;(R) $AFE804...$AFE807
```

Note: This feature is not implemented in the Foenix IDE yet.

```
; SD Card CH376S Port
```

```
SDCARD_DATA    = $AF:E808  ;(R/W) SDCARD (CH376S) Data PORT_A (A0 = 0)
```

```
SDCARD_CMD     = $AF:E809  ;(R/W) SDCARD (CH376S) CMD/STATUS Port (A0 = 1)
```

```
; SD Card Card Presence / Write Protect Status Reg
```

```
SDCARD_STAT    = $AF:E810  ;(R) SDCARD (Bit[0] = CD, Bit[1] = WP)
```

```
; Audio WM8776 CODEC Control Interface (Write Only)
```

```
CODEC_DATA_LO = $AF:E820  ;(W) LSB of Add/Data Reg to Control CODEC See WM8776 Spec
```

```
CODEC_DATA_HI = $AF:E821  ;(W) MSB od Add/Data Reg to Control CODEC See WM8776 Spec
```

```
CODEC_WR_CTRL = $AF:E822  ;(W) Bit[0] = 1 -> Start Writing the CODEC Control Register
```

Note: This feature is not implemented in the Foenix IDE yet.

Communicating with the Machine over the USB Port

The machine has a USB port that allows for easy transfer of data from the C256 IDE directly to the machine.

When sending commands to the machine, an 8-byte array is the minimum length.

- Byte 0: Header = $55
- Byte 1: Command
- Byte 2: Destination Address (H)
- Byte 3: Destination Address (M)
- Byte 4: Destination Address (L)
- Byte 5: Destination Size (H)
- Byte 6: Destination Size (L)
- Byte 7: Checksum – XOR all the previous bytes

The Destination Address and Destination Size only needs to be specified if writing to memory.

Read from Memory Command ($00)

Write to Memory Command ($01)

Program Flash Command ($10)

The command "Program Flash", you need to supply the Pointer in RAM of where the Flash image will be. The overhead for this one is even longer, each write byte in Flash takes about 20us. So 512K x 20us = 10Sec. So, in theory, you should have an answer to that command 10 second after sending it, indicating that it is now complete.

Erase Flash Command ($11)

Now, the Command "Erase Flash" doesn't have any specific parameters, but the command needs to follow the same protocol as "Stop CPU" or "Start CPU". The overhead to receive an answer from that command is around 100ms. So, make sure the serial interface can wait that long.

Set Debug Mode Command ($80)

Exit Debug Mode Command ($81)

Get Revision ($FE)

Use this to receive the Debug core version, if you want to make the difference between RevB2 and other revisions.

Value returned:

- $00 - RevB2
- $01 - Everything at RevC4A and after (or RevC4 with more features)

Sample Code

Use the VDMA in linear mode to transfer data.

Linear Mode VDMA Transfer

```
SETUP_VDMA_FOR_TESTING_1D
        setas
        LDA #$01 ; Start Transfer
        STA @lVDMA_CONTROL_REG

        LDA #$FE
        STA @lVDMA_SIZE_L
        LDA #$9F
        STA @lVDMA_SIZE_M
        LDA #$00
        STA @lVDMA_SIZE_H

        LDA #$64
        STA @lVDMA_DST_ADDY_L
        LDA #$84
        STA @lVDMA_DST_ADDY_M
        LDA #$03
        STA @lVDMA_DST_ADDY_H

        LDA #$55
        STA @lVDMA_BYTE_2_WRITE

        LDA #$85 ; Start Transfer
        STA @lVDMA_CONTROL_REG
        LDA @lVDMA_STATUS_REG
        RTS
```

Block Mode VDMA Transfer

```
SETUP_VDMA_FOR_TESTING_2D
        setas

VDMA_WAIT_TF
        ; Wait for the Previous Transfer to be Finished
```

```
        LDA @lVDMA_STATUS_REG
        AND #VDMA_STAT_VDMA_IPS
        CMP #VDMA_STAT_VDMA_IPS
        BEQ VDMA_WAIT_TF

        LDA #$01 ; Start Transfer
        STA @lVDMA_CONTROL_REG

        LDA #200
        STA @lVDMA_X_SIZE_L
        LDA #00
        STA @lVDMA_X_SIZE_H

        LDA #64
        STA @lVDMA_Y_SIZE_L
        LDA #00
        STA @lVDMA_Y_SIZE_H

        LDA #$60
        STA @lVDMA_DST_ADDY_L
        LDA #$90
        STA @lVDMA_DST_ADDY_M
        LDA #$01
        STA @lVDMA_DST_ADDY_H

        LDA #$80
        STA @lVDMA_DST_STRIDE_L
        LDA #$02
        STA @lVDMA_DST_STRIDE_H

        LDA #$F9
        STA @lVDMA_BYTE_2_WRITE

        LDA #$87 ; Start Transfer
        STA @lVDMA_CONTROL_REG
        LDA @lVDMA_STATUS_REG
        RTS
```

Clear Screen with VDMA

Code Example of Bitmap ClearScreen with the 2D Mode that happens to work better than the 1D Mode:

```
CLEAR_BITMAP
        setas

CLR_SCREEN_CHECK_NO_ACTIVE_DMA
        ; Wait for the Previous Transfer to be Finished
        LDA @lVDMA_STATUS_REG
        AND #VDMA_STAT_VDMA_IPS
        CMP #VDMA_STAT_VDMA_IPS
        BEQ CLR_SCREEN_CHECK_NO_ACTIVE_DMA

        LDA #$01 ; Enable VDMA Block
        STA @lVDMA_CONTROL_REG

        LDA #$80
        STA @lVDMA_X_SIZE_L
        LDA #$02
        STA @lVDMA_X_SIZE_H

        LDA #$E0
        STA @lVDMA_Y_SIZE_L
        LDA #$01
        STA @lVDMA_Y_SIZE_H

        LDA #$00
```

```
STA @lVDMA_DST_ADDY_L
LDA #$00
STA @lVDMA_DST_ADDY_M
LDA #$00
STA @lVDMA_DST_ADDY_H

LDA #$80
STA @lVDMA_DST_STRIDE_L
LDA #$02
STA @lVDMA_DST_STRIDE_H

LDA #$00
STA @lVDMA_BYTE_2_WRITE

LDA #$87 ; Start Transfer
STA @lVDMA_CONTROL_REG
LDA @lVDMA_STATUS_REG
RTS
```