Neo6502 Documentation

Paul Robson

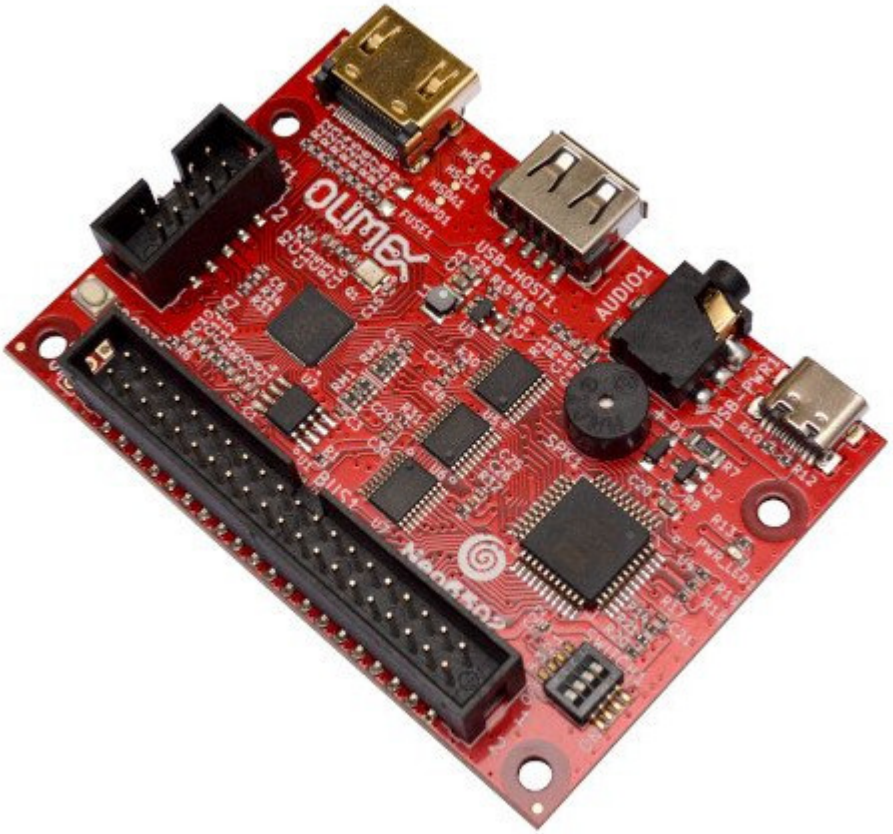
Paul Robson and others

Open Source

Table of contents

1. Welcome to the Olimex Neo6502 Documentation

There is pdf copy of this documentation here

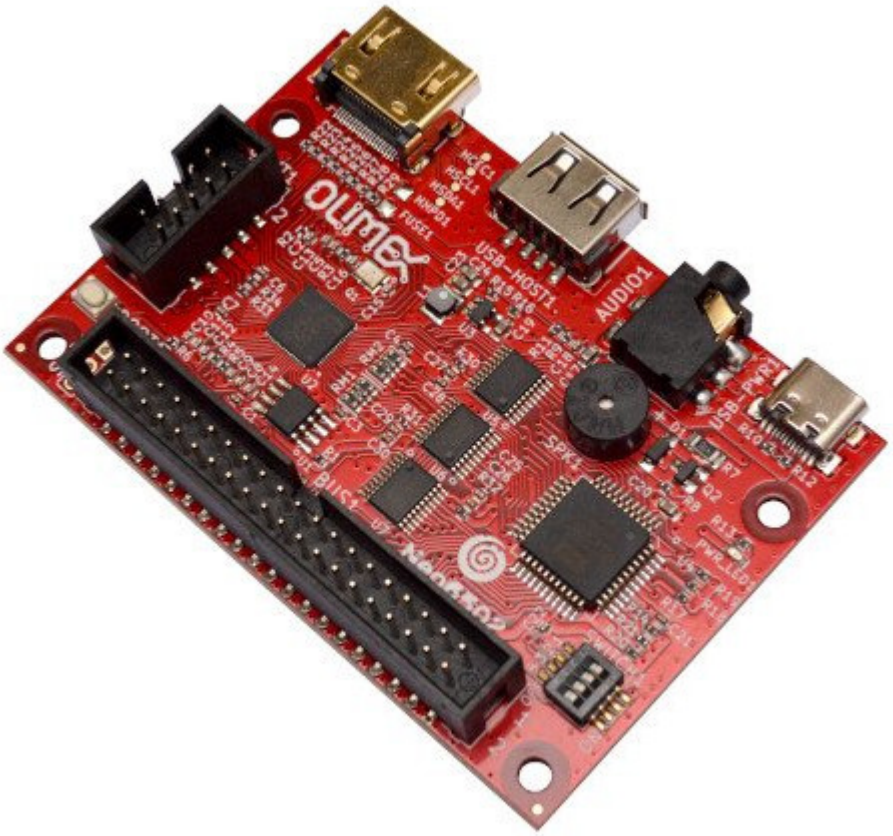

Olimex
USB-HOST1
AUDIO1
SPK1
BUS1

- Real 65C02 Processor clocked at 6.25 Mhz
- "Clean" machine, 64k RAM available
- Low cost purchase, sold for 30 Euros
- All software and hardware is open source
- Wifi support (requires additional inexpensive board)
- 320 x 240 256 colour display on HDMI/DVI with a palette
- 32k Graphics RAM for tiles and sprites
- 128 sprites up to 32x32 pixels.
- Multiple tile maps (16x16 tiles, can be double sized)
- High speed drawing features
- Turtle Graphics
- Blitter for high speed graphics
- UEXT interface to access a wide range of hardware add ons.
- 4 channels of square wave or white noise sound
- Predefined set of sound effects.
- Storage USB Key (optionally can use SD Card)
- Fast structured BASIC with hardware support and inline assembler.
- BASIC can be edited on screen, or using a text editor.
- High Speed Integer/Floating point arithmetic
- Documentation, samples, explainers and games, all open source.
- Cross development support
- Accurate cross platform emulator for Windows/Mac/Linux, only requires SDL2
- Serial link to PC for Cross-Development
- Program in PASCAL using Mad Pascal compiler
- Program in 'C' using CC65 and LLVM
- USB Mouse and Gamepad support
- BASIC support for Serial, I2C and SPI hardware via UEXT Connector

2. Read me first

2.1 What is the Olimex Neo6502 ?

This is the Olimex Neo6502 main board. The main components are a 65C02 and a Raspberry Pi Pico. The 65C02 runs the machine code (at about 6.3Mhz) ; the Pico does pretty much everything else.



Most of this document describes its use for a seperate new Retrocomputer design. There are other projects which use the board to emulate real machines ; currently the Apple 2, Oric and Commodore 64 are in development. At the time of writing I do not believe the C64 emulator is publicly released.

There are other models ; early adopters had an A board (this is version B) which is almost identical, but Purple rather than Red, and requires a couple of wires to work properly.

There is also a 'portable' machine which exists in prototype form (at the time of writing) which has an LCD Screen, 4 UEXT sockets and a built in hub.

[What to think about before buying]

2.2 Read this before Purchasing

2.2.1 What do you need

Besides the board itself, you will need various other devices to make the computer work. Most of these are relatively straightforward.

Some however are **worth acquiring when you buy the board**. This is because some distributors, most in fact, charge a significant amount for postage for small orders, so if you order them later it will be more expensive. The devices shown below, and most of the UEXT boards are excellent value for money.

USB Hub issues.

The Raspberry Pi PICO has a technical problem when used in conjunction with the tinyUSB library. Not all USB hubs work with it. If you want to use the Apple, Oric, or C64 emulators you may not need a USB hub, but "Morpheus" does. Following some investigation, it was discovered that only certain chipsets worked, so Olimex produced a small USB hub using that chipset.

Buying this is well worth considering as it is not expensive (currently 8 Euros). If you have a USB hub, it may be worth trying. The Raspberry Pi hubs built into the keyboard appear to work correctly. But this cannot be guaranteed as the chips used may change.

This has been reported as a fault, investigated, and it seems at present it is not fixable. It is possible this hub may be integrated onto later versions of the board.

Optional Gamepad

Gamepads are easier to acquire. This Gamepad, a copy of a Super Nintendo controller with a USB connector can be acquired from several sources. One of the issues with USB Game controllers is that there seems to be no standard (if you look at the Linux source there is a list of driver options for all sorts of keyboards). Currently we only support certain types of keyboards. Again, this is obtainable from Olimex and it will be the 'correct model'. The author has similar looking gamepads not acquired from Olimex, and they seem to work (it depends on the USB ID) but they may not. There is support for some other gamepads included.

It's not a requirement. The "Operating System" of Morpheus uses keyboard keys if no gamepad is present.

A-A cable or Programmer

To program the Pico , you either need a USB A - USB A cable is required, it is also possible to use a Programmer and the 'openocd' software as developers for the Pico do. The Neo6502 usually comes programmed with an Apple 2 emulator and Morpheus is still updated for bug fixes and occasional enhancements. It is not required for to day to day use of the machine.

Case

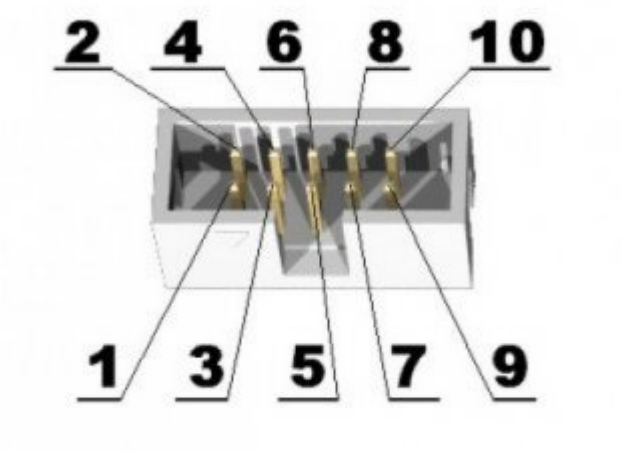
This is completely optional ; it is a small box which the Neo6502 board fits into, all the main sockets and connectors are exposed. For P&P reasons , if you want it, it probably makes sense to order it at the same time as the board. It is available in blue or red lettering.

2.2.2 Internet connection

Neo6502 now has an internet connection and a program source written by Wojciech Bocianski (aka "Bocianu"). This is not built in and requires this board which plugs into the UEXT port.

UEXT Devices

Olimex produced UEXT modules which are listed here, for the same reasons if you want one in particular it is worth ordering it at the same time. The majority of these devices are economically priced. The Basic and API have simple commands for interfacing with SPI, I2C and Serial Devices.



2.2.3 Common Devices

Keyboard

A standard USB keyboard is used, these seem to work almost 100%.

HDMI display

The output is DVI through a HDMI connector. There is currently a discussion about whether it should be powered or not, currently it is not. It does not seem to work well with adaptors and some displays, notably LG.

USB Key

If you want to save programs then a USB key is required (It is possible to use SD Cards). For similar reasons to the USB Hub we believe, it requires a fast key. Initially I used "Amazon Cheapies" and none of them worked properly. I replaced this with a Sandisk USB 3.0 key which was about £10 and it works fine.

Power Cable

Power is supplied through a USB C type cable of the type that are commonly available.

Sound Device

Sound is provided on board by a simple buzzer. This is perfectly okay for beeps and squawks, and fairly audible. It is possible to plug a sound device in which has a 3.5mm Jack plug.

SD Card (as an option)

The system does still support the use of SD Cards. This requires the Olimex UEXT SD Card adaptor.

Where to buy a Neo6502

2.3 Where can you buy a Neo6502 ?

Missing suppliers who wish to be added contact me on Discord

2.3.1 Olimex (Europe and worldwide)

The home of the Neo6502 is Plovdiv in Bulgaria, and all the hardware can be bought from Olimex, who deliver worldwide.

2.3.2 Authorised Resellers

The above resell locally on behalf of Olimex.

Mouser (Worldwide)

Mouser stock Neo6502s for next day delivery.

The Pi Hut (UK)

For UK residents the Pi Hut is an option.

Agon Australia (Australia ... unsurprisingly)

Supporters of the truly *dreadful* Agon Light (note: this is not serious, it's a great piece of hardware) have come to their senses and have started stocking the Neo6502.

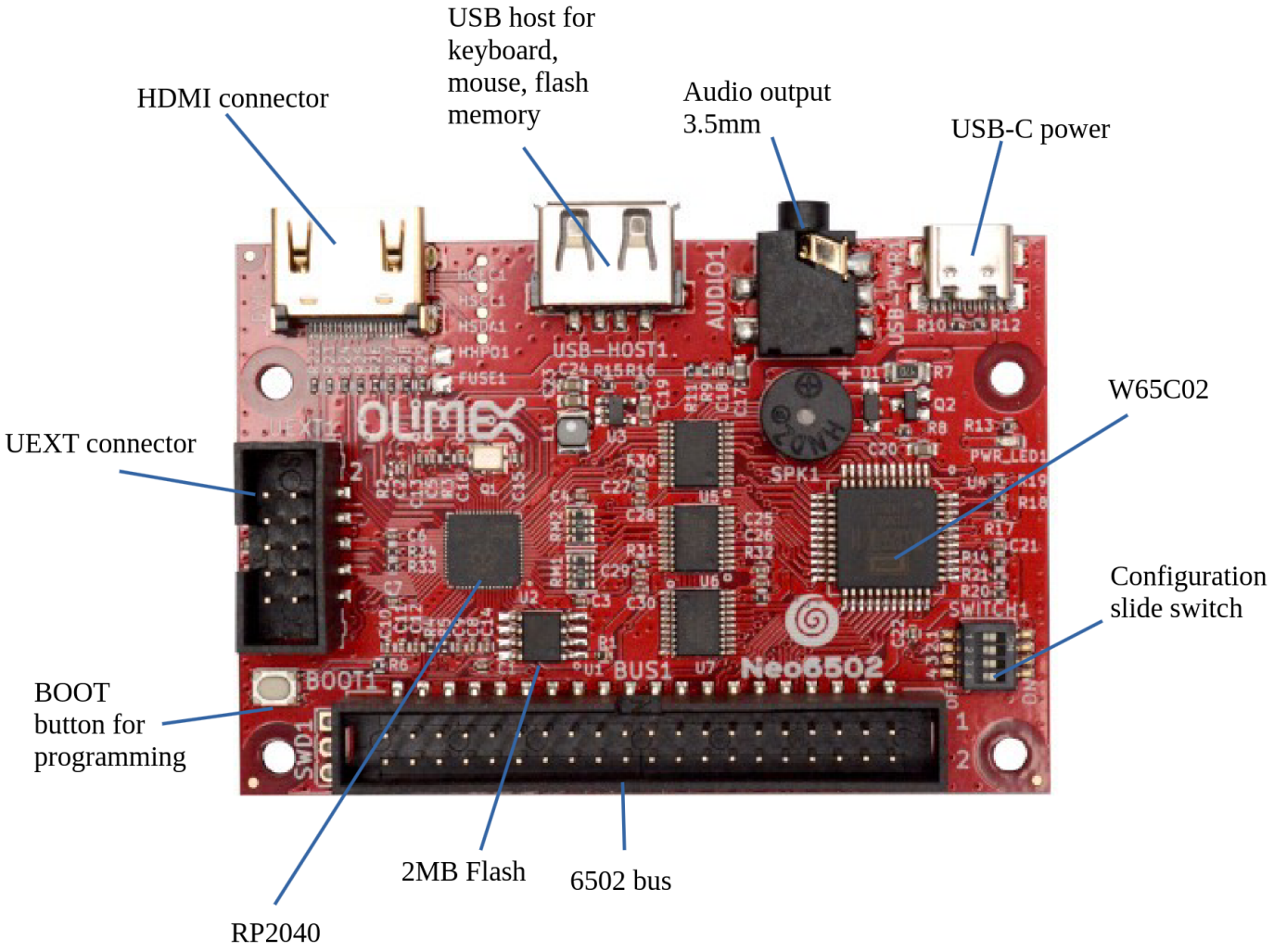
DigiKey (Worldwide)

Digikey stock Neo6502s for next day delivery

What to do "out of the box"

2.4 Getting the board running

This section helps you get the board up and running with the Morpheus software. If you want to use the Apple, Oric or C64 emulators, the process is similar, but you use a different UF2 file.



2.4.1 Setting the DIP switches

There is a small configuration slide switch next to the long connector on one side of the board, marked "SWITCH1". Olimex supplies these as "all on" (as per the diagram above).

Switch 1 enables and disables the buzzer. Switches 2-4 connect UEXT lines to NMI, IRQ and Reset and it is strongly recommended that these are set to off.

2.4.2 USB Key

The USB key should be formatted to FAT32

2.4.3 Wiring it up

1. Plug the HDMI Monitor into the HDMI connector using a standard HDMI cable.
2. Plug the USB hub, if you are using one, or the keyboard, into the USB host socket and plug the Gamepad, USB Key and Keyboard into the USB host
3. Plug the power cable into the USB C socket, and connect the other end to power.

This *should* boot up the Apple 2 emulator, which should appear in the form of a menu of games, which you can select and run using the keyboard.

If not, try removing the USB host and just plugging the keyboard into the middle socket.

If this doesn't work, try reprogramming it with Morpheus (see below) and see what happens there.

Note for SD Card users

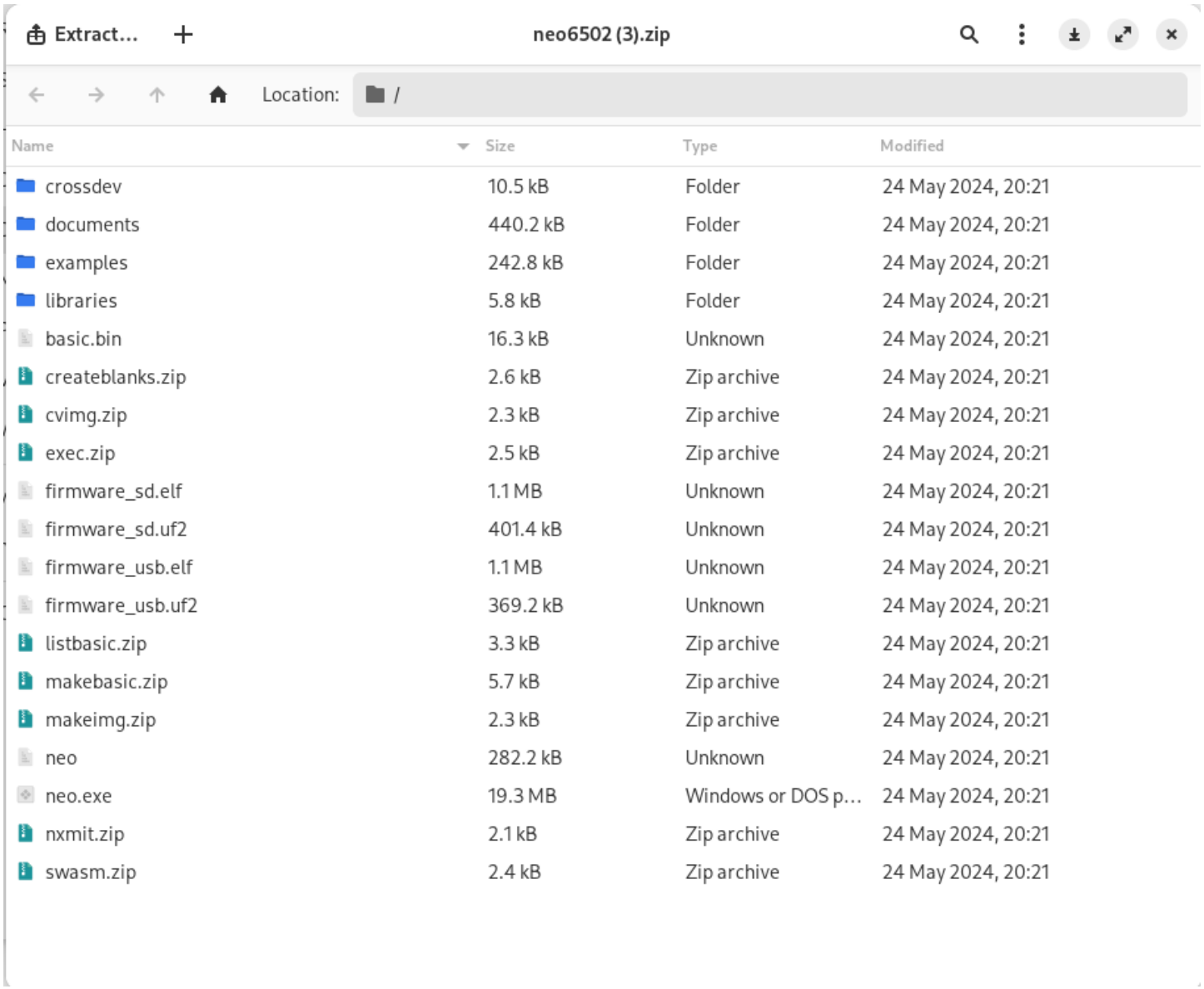
If you are using the SD card it should come with a short ribbon cable which plugs into the SDCard board and the UEXT socket on the Neo6502 board.

2.4.4 Reprogramming with Morpheus

First you need to download the current release of the Morpheus software here .

The releases are on the right of the screen, the current release is 0.30.0. Clicking on it should show the releases page which has a link to "neo6502.zip"

Download and extract this file. You will see something like this, this is Linux, but it will be very similar on Windows or Macintosh.

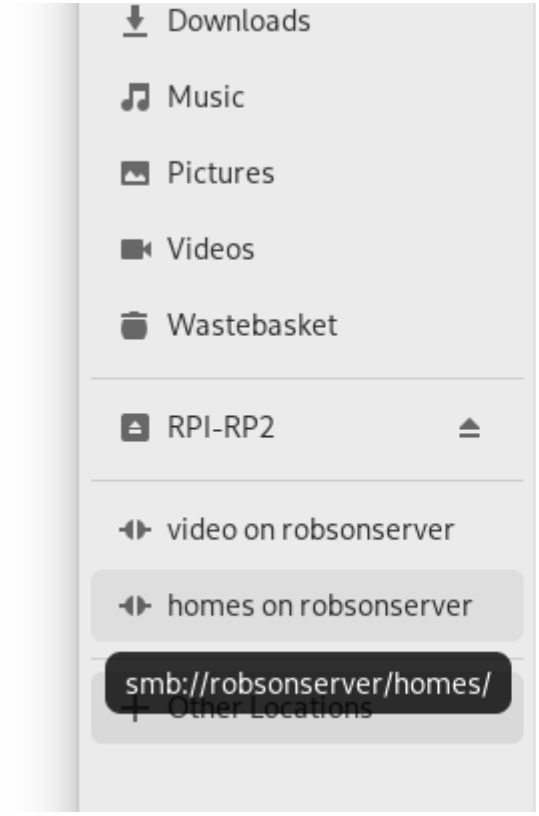


Initially you need one of the two files in the middle "firmware_usb.uf2" or "firmware_sd.uf2". Most people will need the first one, the second one is if you use the SD Card for storage.

You can also copy the examples/basic files to your storage device (normally the USB key). It is recommended you create one file just for testing.

1. Remove the keyboard or USB hub from the middle socket, and connect the Neo6502 board to your PC using the A-A cable - this should fit into any standard PC socket.
2. Turn the power to the board off.
3. You now need to put the board into "Upload" mode. This is a similar upload to any other Pi Pico device. Hold the boot button down (see the picture above) and turn the power back on, then release the boot button. If you have already put the board in the case there is a hole to access the boot button which needs something thin and pointy (I use an old multimeter probe)

The board should now appear in the file system of your host computer. For example, on Linux/Gnome it is like this. Exactly what you see will depend on your operating system and desktop, but you should see "RPI-RP2" mounted as a drive.



You now need to copy the uf2 file above to this drive. This again depends on the system you are using.

Hopefully the system should now boot. The machine should make it's boot sound (a low beep and a high beep - it sounds like a BBC Micro) listen here and it should display the Neo6502 logo, title, and Olimex title.

Finally unplug the A cable and plug the USB hub or keyboard back in. More than once I have forgotten to do this and wondered why the keyboard is not working.

2.4.5 What does the boot mean

Below the logos, you should see.

Morpheus Version

The Morpheus title and version in Yellow - it currently says "Morpheus Firmware: v 0.28.1"

Storage Type

It should say "USB Storage" or "SD Card Storage" depending on your choice of firmware

USB Devices

Your USB devices should now be listed. Mine says:

- No driver found for 04D9:0006 (this is the Pi Pico USB hub I think)
- USB Key found 0781:55a3
- Gamepad driver found 081f E401

Not all devices are detected, this does not matter.

Checking it works and a look round

2.4.6 Check it works and doing a Catalogue

Below the USB list it should say "Welcome to NeoBasic" in green. You are now running a classic BASIC interpreter - so for example

`print 22/7` [ENTER]

causes 3.142857 to be printed. If you type

`cat`

it should list all the files in the root directory of the USB key

2.4.7 Running a simple BASIC program

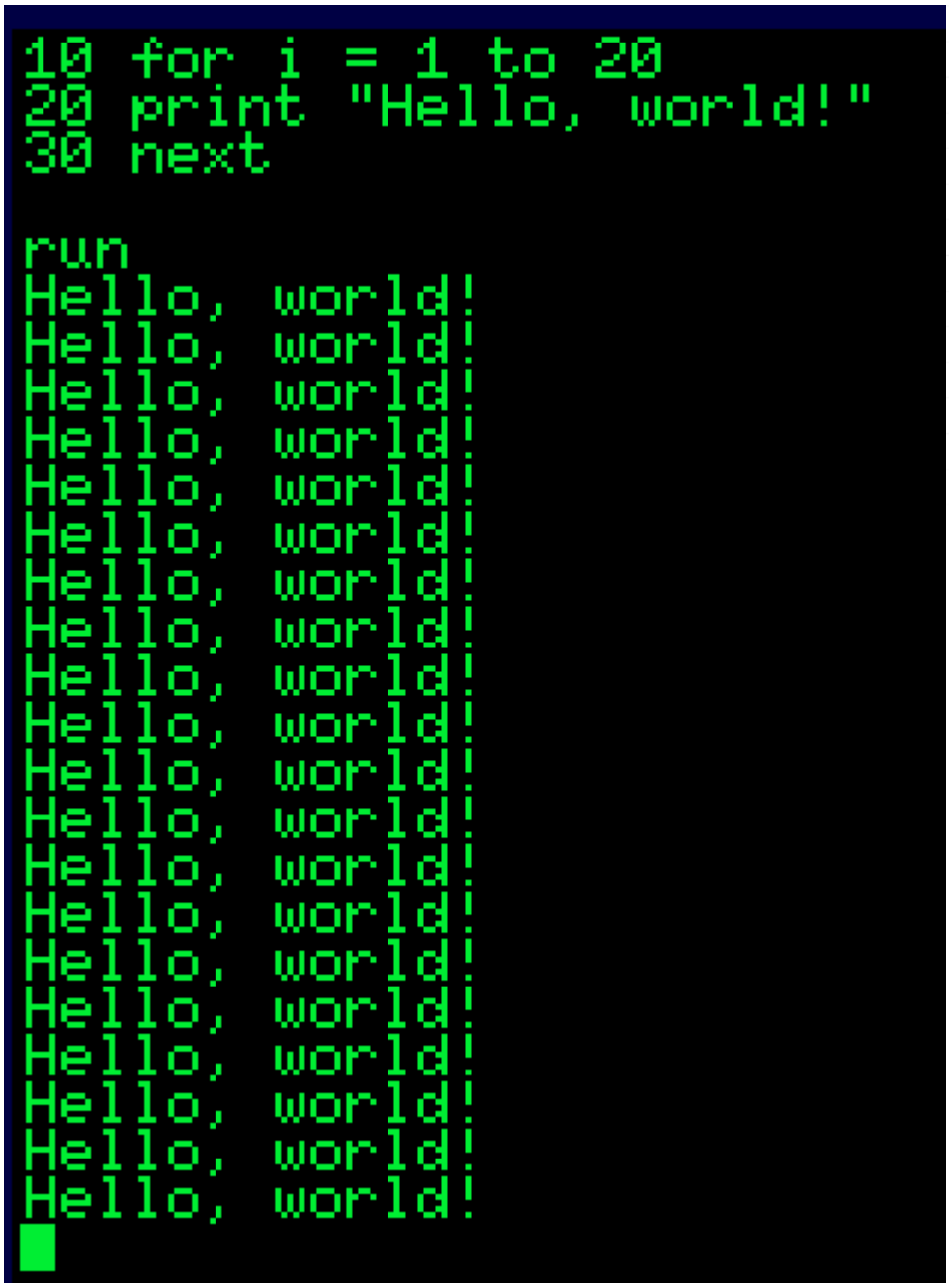
To get you started, and show you some of our neater features, we will write a very simple BASIC program, which many of you will have seen many times.

```
10 for i = 1 to 10

20 print "Hello, world !"

30 next
```

and type run to run it, like this.

Using the screen editor

Many people don't like line number programming. Even with an on screen editor - you can move the arrow keys up and make the 10 at the end of line 10 to 20, press [ENTER] and run it again.



Using the BASIC Text Editor

But the developers decided it would be better to have a proper editor. (It is possible to cross-develop in BASIC using the emulator or the serial port)

So type `edit` and press ENTER

And you can edit like on a word processor (Ctrl+P and Ctrl+Q insert/delete lines). Not a very good word processor, but it's better than line numbers.

When you have finished press ESC to leave the editor and you can run it again. You can speed this up using the function keys e,g, from the BASIC command line (e.g. where you type stuff in)

```
fkey 1,"edit"+chr$(13)
```

This sets function key F1 on the keyboard to type edit and press ENTER (character code 13).

Saving Loading and "Cat"ting

You can save your program with

```
save "hello.bas"
```

It is a convention that .bas is used for BASIC files, but it's not mandatory. You can reload it with `load "hello.bas"` and run it with `run "hello.bas"`

`cat` should show it stored in the directory ; if you have lots of files already you can `cat "hell"` which will show all files with hell in it.

[Where do I go next ?]

2.5 Where Next

So you now have a working Neo6502. Here are some places to explore.

2.5.1 Discord

There is an active discord where you can ask questions or whatever of the people involved in software and hardware, and ask for assistance.

There is a link to join it here

2.5.2 Languages available

C and Pascal development are currently working well (both LLVM and CC65) as well as Assembler and BASIC. Forth and a variant are currently under development. There is an early version of LLVM documentation and a more complete Pascal example on the wiki page.

2.5.3 Other software

Other software is available, linked from the Wiki, now including a Tile map and Sprite editor, and a file manager, and some games.

2.5.4 Facebook

There is a facebook group here

2.5.5 Assembler, BASIC , C and Pascal examples

Under "examples" are a variety of sample programs some of which are full programs, some demonstrate specific features of BASIC or the API. The .bsc files are text versions of the BASIC files.

2.5.6 The Youtube series

There is a 7 part youtube video series of about 30 minutes which walks through programming a simple game in BASIC using cross development and external tools. Much of it is applicable to C and Pascal which uses the same API - many BASIC commands are simple wrappers round the API functions.

3. Reference

3.1 Neo6502 Messaging API

The Neo6502 API is a messaging system. There are no methods to access the hardware directly. Messages are passed via the block of memory from $FF00 to $FF0F, as specified in the "API Messaging Addresses" table on the following page.

The kernel include file documents/release/neo6502.inc specifies the beginning of this address range as the identifier ControlPort, along with the addresses of WaitMessage and SendMessage (described later), various related kernel jump vectors, and some helper functions.

The application include files examples/assembly/neo6502.asm.inc and examples/C/neo6502.h also specify the beginning of this address range as the identifier ControlPort. The assembly include also specifies ControlPort and the other controls as API_COMMAND, API_FUNCTION, API_ERROR, API_STATUS, and API_PARAMETERS. The C header also specifies ControlPort and the other controls as API_COMMAND_ADDR, API_FUNCTION_ADDR, API_ERROR_ADDR, API_STATUS_ADDR, and API_PARAMETERS_ADDR.

API Commands/Functions are grouped by functionality. For example, Group 1 are system-related, and Group 2 are console-I/O-related.

Command/Function Parameters are notated in this document as Params[0] through Params[7], or as a list or range (eg: Params[1,2], Params[0..7]). Note that these are referring to a mapping to memory locations. The numbers represent offsets from the Parameters base address $FF04. Ie: the actual bytes are not necessarily all distinct "parameters" in the conventional sense. Depending on the routine, a logical parameter may be an individual byte, one or more bits of a byte interpreted as a composite or bit-field, or multiple adjacent bytes interpreted as 16 or 32 bit values. For example, the list Params[0,1] would indicate a single logical parameter, comprised of the two adjacent bytes $FF04 and $FF05. The range Params[4..7] would indicate a single logical parameter, spanning consecutive bytes between $FF08 and $FF0B.

Note that strings referenced by Parameters are not ASCIIZ, but are length-prefixed.

The first byte represents the length of the string (not counting itself). The string begins at the second byte. Consequently, strings must be 255 bytes or less (not counting the length header).

API Messaging Addresses

Address	Type	Notes
FF00	Group	Group selector and status. Writing a non-zero value to this location triggers the routine specified in $FF01. The system will respond by setting the 'Error', 'Information', and 'Parameters' values appropriately. Upon completion, this memory location will be will cleared.
FF01	Function	A command or function within the selected Group. For example, Group 1 Function 0 writes a value to the console; and Group 1 Function 1 reads the keyboard.
FF02	Error	Return any error values, 0 = no error.
FF03:7	Status	Set (1) if the ESCape key has been pressed. This is not automatically reset.
FF03:6	*unused*	
FF03:5	*unused*	
FF03:4	*unused*	
FF03:3	*unused*	
FF03:2	*unused*	
FF03:1	*unused*	
FF03:0	*unused*	
FF04..	Parameters	This memory block is notated in this document as Params[0] through Params[7], or as a composite list or range (eg: Params[1,2], Params[0..7]). Some Functions require Parameters in these locations and some return values in these locations; yet others do neither.

3.1.1 API Interfacing Protocol

Neo6502 application programmers should interact with the API per the following algorithm:

1. Wait for any pending command to complete. There is a subroutine WaitMessage which does this for the developer.
2. Setup the Function code at $FF01; and any Parameters across $FF04..$FF0B. To print a character for example, set $FF01 to $06 and set $FF04 to the character's ASCII value. To draw a line, set $FF01 to $02 and set $FF04..$FF0B as four 16-bit pixel coordinates:
3. Setup the command code at $FF00. This triggers the routine; so mind that the Function code and Parameters are setup sanely prior. On a technical point, both implementations process the message immediately on write.
4. Optionally, wait for completion. Most commands (eg: writing to the console) do not require waiting, as any subsequent command will wait/queue as per step 1. Query commands (e.g. reading from the keyboard queue), return a value in a parameter. Programs must wait until the control address $FF00 has been cleared before reading the result of a query.

There is a support function SendMessage, which in-lines the command and function. E.g.: this code from the Kernel:

```
jsr KSendMessage ; send message for command 2,1 (read keyboard)

.byte 2,1

jsr KWaitMessage ; wait to receive the result message

lda DParameters ; read result
```

The instructions above are equivalent to the following explicit steps:

```
lda #1

sta DFunction

lda #2

sta DCommand ; trigger API function 2,1 (read keyboard)

Loop:

lda DCommand ; load the result - non-zero until the routine's completion

bne Loop ; wait for API routine to complete

lda DParameters ; read result (a key-code)
```

3.1.2 Mathematical Interface

The mathematical interface of the API functions largely as a helper system for the BASIC interpreted, but it is open to any developer who wishes to avail themselves of the functionality. It is strongly advised that function 32 of Group 4 (NumberToDecimal) is not used as this is extremely specific and as such is not documented.

The interface is used in a stack environment, but is designed so it could be used in either a stack environment or a fixed location environment. The Neo6502 BASIC stack is also 'split', so elements are not consecutive, though they can be.

Parameter 0 and 1 specify the address of the registers 1 and 2. Register 1 starts at this address, Register 2 starts at the next address. Parameter 2 specifies the step to the next register. Therefore they are interleaved by default at present.

So if Parameters 0 and 1 are 8100 and Parameter 2 is 4, the 5 byte registers are

- Register 1: 8100,8104,8108,810C,8110
- Register 2: 8101,8105,8109,810D,8111

Bytes 1-4 of the 'register' are the number, which can be either an integer (32 bit signed) or a standard 'C' float (e.g. the IEEE Single Precision Float format). Bit 0 is the type byte, and the relevant bit is bit 6, which is set to indicate bytes 1-4 are a float value, and is set on return.

Binary functions that use int and float combined (one is int and one is float) normally return a float.

3.2 Console Codes

The following are console key codes. They can be printed in BASIC programs using chr$(n), and are also related to the character keys returned by inkey$(). The key() function uses physical key numbers. Some control codes do not have corresponding keyboard keys; and some console outputs are not yet implemented.

Backspace (8), Tab (9), Enter/CR (13), Escape (27), and the printable characters (32..127) are the standard ASCII set. Other typical control keys (eg: Home and arrows) are mapped into the 0..31 range.

Console Key Codes - Non-Printable

Code	Ctrl	Key	Function
1	A	Left Arrow	Cursor Left
4	D	Right Arrow	Cursor Right
5	E	Insert	Insertion Mode
6	F	Page Down	Cursor Page Down
7	G	End	Cursor Line End
8	H	Backspace	Delete Character Left
9	I	Tab	Tab Character
10	J		Line Feed
12	L		Clear Screen
13	M	Enter	Carriage Return (Accept Line)
18	R	Page Up	Cursor Page Up
19	S	Down	Cursor Down
20	T	Home	Cursor Line Begin
22	V		Cursor Down (8 Lines)
23	W	Up	Cursor Up
24	X		Cursor Color Inverse
26	Z	Delete	Delete Character Right
27	[	Escape	Exit

Console Key Codes - Printable

Code	Type	Notes
20-7F	ASCII Set	Standard ASCII Characters
80-8F		Set Foreground Color
90-9F		Set Background Color
C0-FF		User-definable Characters

3.3 Graphics

3.3.1 Graphics Settings

Function 5,1 configures the global graphics system settings. Not all Parameters are relevant for all graphics commands; but all Parameters will be set by this command. So mind their values.

The actual color of each drawn pixel will be computed at runtime by AND'ing the existing pixel color with the value specified in Params[0], then XOR'ing the result with the value specified in Params[1].

The value in Params[2] is a flag which determines the paint fill mode for the Draw Rectangle and Draw Ellipse commands: reset (0) for outline, set (1) for solid fill.

The value in Params[3] is the draw extent (window) for the Draw Image command.

The value in Params[4] is a bit-field of flags for the Draw Image command, which determine if the image will be inverted (flipped) horizontally or vertically: bit-0 for horizontal, bit-1 for vertical, reset (0) for normal, set (1) for inverted.

For the "Draw Rectangle" and "Draw Ellipse" commands, the given order and position of the coordinates are not significant. To be precise, one is "a corner" and the other is "the opposite corner". For the "Draw Ellipse" command, these corners are referring to the bounding-box. The coordinates for an ellipse will lie outside of the ellipse itself.

3.3.2 Graphics Data

Graphics data files are conventionally named ending in the .gfx suffix; though this is not mandatory. The format is quite simple.

Graphics Data Format

Offset	Data	Notes
0	1	Graphics Data Format ID
1	Count	Number of 16x16 tiles in use
2	Count	Number of 16x16 sprites in use
3	Count	Number of 32x32 sprites in use
4..255		Reserved
256	Raw	Sprites graphics data

The layout of sprites graphics data is all of the 16x16 tiles, followed by all the 16x16 sprites, followed by all the 32x32 sprites, all in their respective orders. As there is currently only about 20kB of Graphics Memory, these should be used somewhat sparingly.

Each byte specifies 2 pixels. The upper 4 bits represent the first pixel colour; and the lower 4 bits represent the second pixel colour. So tiles and 16x16 sprites occupy 16x16/2 bytes (128 bytes) each. Each 32x32 sprite occupies 32x32/2 bytes (512 bytes). Colour 0 is transparent for sprites (colour 9 should be used for a black pixel).

The release package includes Python scripts for creating graphics files, which allow you to design graphics using your preferred editing tools (eg: Gimp, Inkscape, Krita, etc). There is an example in the crossdev directory, which demonstrates how to get started importing graphics into the Neo6502.

3.4 Sprites

The Neo6502 graphics system has one sprite layer (z-plane) in the conventional sense. Technically, there is no "sprite layer", per-se. The system uses palette manipulation to create, what is in practice, a pair of 4-bit bit-planes. The sprite graphics are in the upper nibble, the background is in the lower nibble, and the background is drawn only if the sprite graphic layer is zero. It's this top nibble which is read by Function 5,36 "Read Sprite Pixel".

Function 6,2 sets or updates a sprite. These parameters (eg: the X and Y coordinates) cannot be set independently. To retain/reuse the current value of a parameter for a subsequent call, set each of the associated byte(s) to $80 (eg: $80,$80,$80,$80 for coordinates).

The 'Sprite' Parameter Params[0] specifies the index of the sprite in the graphics system. Sprite 0 is the turtle sprite.

Params[1,2],Params[3,4] specifies the X and Y screen coordinates.

Bits 0-5 of the 'Image' Parameter Params[5] specify the index of a specific graphic within the sprites data. Bit 6 of the 'Image' Parameter specifies the sprite dimensions: reset (0) for 16x16, set (1) for 32x32. In practice, the index is the same as the sprite number ($80-$BF for 16x16 sprites, $C0-$FF for 32x32 sprites), but without bit-7 set.

The value in Params[6] specifies a bit-field of flags, which determines if the graphic will be inverted (flipped) horizontally or vertically: bit-0 for horizontal, bit-1 for vertical, reset (0) for normal, set (1) for inverted.

Params[7] specifies the anchor alignment.

3.4.1 Sprite Anchors

The table below shows the valid anchor alignments for a sprite. The anchor position is the origin of the relative coordinate given. That is, coordinates 0,0 of the sprite will coincide with one of the positions shown in the table below. The default anchor alignment is zero (middle-center).

7	8	9
4	0/5	6
1	2	3

To the right are two examples. Assume this is a 32x32 sprite. In the upper example, the anchor point is at 8, the top-center. Considering the origin at the bottom-left, this sprite is drawn at 16,32, the midpoint of the top of the square.

In the lower example, the anchor point is at 0; and this sprite is drawn at 16,16 (the middle of the square). The anchor point should be something like the centre point. So for a walking character, this might be anchor point 2 (the bottom-center).

3.5 Sound

Function 8,4 queues a sound. Queued sounds are played sequentially, each after the previous has completed, such that sounds within a channel queue will not conflict, interrupt, or overlap.

Frequency is in units of Hertz. Duration is in units of 100ths of a second. Slide is a gradual linear change in frequency, in units of Hz per 100th of a second. Sound target type 0 is the beeper. Currently, the beeper is the only available sound target.

Queue Sound Parameters

FF04	FF05	FF06	FF07	FF08	FF09	FF0A	FF0B
Channel	Freq Low	Freq High	Length Low	Length High	Slide Low	Slide High	Target

Function 8,7 is an extended version of this ; function 8.4 still works, but 8,7 allows access to the extended multi channel sound functionality. Sound types are, at the time of writing, 0 (square wave) and 1 (random noise). There are four channels minimum, but the function 8,8 allows you to read the number of supported channels.

FF04	FF05	FF06	FF07	FF08	FF09	FF0A	FF0B	FF0C
Channel	Freq Low	Freq High	Length Low	Length High	Slide Low	Slide High	Sound Type	Sound Volume

3.6 Sound Effects

Function 8,5 plays a sound effect immediately. These will be synthesised to the best ability of the available hardware, so the actual sound may vary slightly. These are predefined as:

ID	Sound
0	positive
1	negative
2	error
3	confirm
4	reject
5	sweep
6	coin
7	las70
8	powerup
9	victory
10	defeat
11	fanfare
12	alarm 1
13	alarm 2
14	alarm 3
15	ringtone 1
16	ringtone 2
17	ringtone 3
18	danger
19	expl100
20	expl50
21	expl20
22	las30
23	las10

3.7 Filing system

The Neo6502 supports the FAT32 file system on a USB mass storage device or SD card. Only the first partition is supported, and filenames must be in the standard DOS 8.3 format.

There is a fairly standard set of functions for loading and saving files, manipulating files and directories, and performing random-access file operations.

3.7.1 File attributes

Files may have any combination of the following bits set.

Name	Value	Meaning
`FIOATTR_DIR`	0x01	This is a directory (may not be modified)
`FIOATTR_SYSTEM`	0x02	This is a system file and will be hidden from directory listings
`FIOATTR_ARCHIVE`	0x04	File is archived; automatically cleared when the file is modified
`FIOATTR_READONLY`	0x08	File is read only and may not be overwritten or modified
`FIOATTR_HIDDEN`	0x10	This will be hidden from directory listings

3.7.2 Error codes

Most of the functions in group 3 will return an error/status code to indicate whether the operation succeeded or not. This will be a value from the following enumeration.

Values here are grouped by their top four bits; applications will typically only need to check the group value unless they are interested in the specific error.

Group 0; miscellaneous

Name	Value	Meaning
`FIOERROR_OK`	0x00	Operation succeeded (not an error)
`FIOERROR_UNKNOWN`	0x01	Something went wrong, but we don't know what
`FIOERROR_EOF`	0x02	A read or directory enumeration operation reached the end of the file
`FIOERROR_UNIMPLEMENTED`	0x03	Operation is not implemented

Group 1: path errors

Name	Value	Meaning
`FIOERROR_NO_FILE`	0x11	Could not find the file
`FIOERROR_NO_PATH`	0x12	Could not find the path
`FIOERROR_INVALID_DRIVE`	0x13	The logical drive number is invalid
`FIOERROR_INVALID_NAME`	0x14	The path name format is invalid
`FIOERROR_INVALID_PARAMETER`	0x15	Given parameter is invalid

Group 2: access errors

Name	Value	Meaning
`FIOERROR_DENIED`	0x21	Access denied due to prohibited access or disk or directory full
`FIOERROR_EXIST`	0x22	Access denied due to prohibited access
`FIOERROR_INVALID_OBJECT`	0x23	The file/directory object is invalid
`FIOERROR_WRITE_PROTECTED`	0x24	The physical drive is write protected
`FIOERROR_LOCKED`	0x25	File is in use

Group 3: media errors

Name	Value	Meaning
`FIOERROR_DISK_ERR`	0x31	A hard error occurred in the low level disk I/O layer
`FIOERROR_INT_ERR`	0x32	Assertion failed
`FIOERROR_NOT_READY`	0x33	The physical drive cannot work
`FIOERROR_NOT_ENABLED`	0x34	The volume has no work area
`FIOERROR_NO_FILESYSTEM`	0x35	The filesystem is invalid

Group 4: internal errors

These indicate an internal error with the Morpheus firmware and should never happen.

Name	Value	Meaning
`FIOERROR_MKFS_ABORTED`	0x41	The f_mkfs() aborted due to any problem
`FIOERROR_TIMEOUT`	0x42	Could not get a grant to access the volume within defined period
`FIOERROR_NOT_ENOUGH_CORE`	0x43	LFN working buffer could not be allocated
`FIOERROR_TOO_MANY_OPEN_FILES`	0x44	fatfs has seen too many open files

3.8 API Functions

The following tables are a comprehensive list of all supported API functions.

For the convenience of application programmers, the application include files examples/C/neo6502.h and examples/assembly/neo6502.asm.inc define macros for these groups, their functions, and common parameters (colors, musical notes, etc).

3.9 Group 1 : System

Function 0 : DSP Reset

Resets the messaging system and component systems. Normally, should not be used.

Function 1 : Timer

Deposit the value (32-bits) of the 100Hz system timer into Parameters:0..3.

Function 2 : Key Status

Deposits the state of the specified keyboard key into Parameter:0.

State of keyboard modifiers (Shift/Ctrl/Alt/Meta) is returned in Parameter:1.

The key which to query is specified in Parameter:0.

Function 3 : Basic

Loads and allows the execution of BASIC via a indirect jump through address zero.

Function 4 : Credits

Print the Neo6502 project contributors (stored in flash memory).

Function 5 : Serial Status

Check the serial port to see if there is a data transmission.

Function 6 : Locale

Set the locale code specified in Parameters:0,1 as upper-case ASCII letters.

Parameter:0 takes the first letter and Parameter:1 takes the second letter.

For example: French (FR) would require Parametr 0 being $46 and Parameter 1 being $52

Function 7 : System Reset

System Reset. This is a full hardware reset. It resets the RP2040 using the Watchdog timer, and this also resets the 65C02.

Function 8 : MOS

Do a MOS command (a '* command') these are specified in the Wiki as they will be steadily expanded.

Function 10 : Write character to debug

Writes a single character to the debug port (the UART on the Pico, or stderr on the emulator). This allows maximum flexibility.

Function 11 : Return Version Information

Reads the current version Major.Minor.Patch into Parameters:0..2 These values are guaranteed to be in the range 0.255

3.10 Group 2 : Console

Function 0 : Write Character

Console out (duplicate of Function 6 for backward compatibility).

Function 1 : Read Character

Read and remove a key press from the keyboard queue into Parameter:0. This is the ASCII value of the keystroke.

If there are no key presses in the queue, Parameter:0 will be zero.

Note that this Function is best for text input, but not for games. Function 7,1 is more optimal for games, as this only detects key presses, you cannot check whether the key is currently down or not.

Function 2 : Console Status

Check to see if the keyboard queue is empty. If it is, Parameter:0 will be $FF, otherwise it will be $00

Function 3 : Read Line

Input the current line below the cursor into Parameters:0,1 as a length-prefixed string; and move the cursor to the line below. Handles multiple-line input.

Function 4 : Define Hotkey

Define the function key F1..F10 specified in Parameter:0 as 1..10 to emit the length-prefixed string stored at the memory location specified in Parameters:2,3.

F11 and F12 cannot currently be defined.

Function 5 : Define Character

Define a font character specified in Parameter:0 within the range of 192..255.

Fill bits 0..5 (columns) of Parameters:1..7 (rows) with the character bitmap.

Function 6 : Write Character

Write the character specified in Parameter:0 to the console at the cursor position.

Refer to Section "Console Codes" for details.

Function 7 : Set Cursor Pos

Move the cursor to the screen character cell Parameter:0 (X), Parameter:1 (Y).

Function 8 : List Hotkeys

Display the current function key definitions.

Function 9 : Screen Size

Returns the console size in characters, in Parameter:0 (height) and Parameter:1 (width).

Function 10 : Insert Line

This is a support function which inserts a blank line in the console and should not be used.

Function 11 : Delete Line

This is a support function which deletes a line in the console and should not be used.

Function 12 : Clear Screen

Clears the screen.

Function 13 : Cursor Position

Returns the current screen character cell of the cursor in Parameter:0 (X), Parameter:1 (Y).

Function 14 : Clear Region

Erase all characters within the rectangular region specified in Parameters:0,1 (begin X,Y) and Parameters:2,3 (end X,Y).

Function 15 : Set Text Color

Sets the foreground colour to Parameter:0 and the background colour to Parameter:1.

Function 16 : Cursor Inverse

Toggles the cursor colour between normal and inverse (ie: swaps FG and BG colors). This should not be used.

Function 17 : Tab() implementation

Internal helper.

Function 18 : Read Ink/Paper Colours

Read the ink/paper colours into Param[0] and Param[1]

Function 19 : Show/Hide Cursor Reversing

Set the cursor visibility to Param[0]. This is reset by clearing the screen.

3.11 Group 3 : File I/O

Function 1 : List Directory

Display the file listing of the present directory.

Function 2 : Load File

Load a file by name into memory. On input:

Parameters:0,1 points to the length-prefixed filename string;

Parameters:2,3 contains the location to write the data to. If the address is $FFFF, the file will instead be loaded into the graphics working memory, used for sprites, tiles, images.

On output:

Error location contains an error/status code.

Function 3 : Store File

Saves data in memory to a file. On input:

Parameters:0,1 points to the length-prefixed filename string;

Parameters:2,3 contains the location to read data from;

Parameters:4,5 specified the number of bytes to store.

On output:

Error location contains an error/status code.

Function 4 : File Open

Opens a file into a specific channel. On input:

Parameter:0 contains the file channel to open;

Parameters:1,2 points to the length-prefixed filename string;

Parameter:3 contains the open mode. See below.

Valid open modes are:

0 opens the file for read-only access;

1 opens the file for write-only access;

2 opens the file for read-write access;

3 creates the file if it doesn't already exist, truncates it if it does, and opens the file for read-write access.

Modes 0 to 2 will fail if the file does not already exist. If the channel is already open, the call fails. Opening the same file more than once on different channels has undefined behaviour, and is not recommended.

Function 5 : File Close

Closes a particular channel. On input:

Parameter:0 contains the file channel to close. If this is $FF this closes all open files.

Function 6 : File Seek

Seeks the file opened on a particular channel to a location. On input:

Parameter:0 contains the file channel to operate on;

Parameters:1..4 contains the file location.

You can seek beyond the end of a file to extend the file. However, whether the file size changes when the seek happens, or when you perform the write is undefined behavior.

Function 7 : File Tell

Returns the current seek location for the file opened on a particular channel. On input:

Parameter:0 contains the file channel to operate on.

On output:

Parameters:1..4 contains the seek location within the file.

Function 8 : File Read

Reads data from an opened file. On input:

Parameter:0 contains the file channel to operate on.

Parameters:1,2 points to the destination in memory,

or $FFFF to read into graphics memory.

Parameters:3,4 contains the amount of data to read.

On output:

Parameters:3,4 is updated to contain the amount of data actually read.

Data is read from the current seek position, which is advanced after the read.

Function 9 : File Write

Writes data to an opened file. On input:

Parameter:0 contains the file channel to operate on;

Parameters:1,2 points to the data in memory;

Parameters:3,4 contains the amount of data to write.

On output:

Parameters:3,4 is updated to contain the amount of data actually written.

Data is written to the current seek position, which is advanced after the write.

Function 10 : File Size

Returns the current size of an opened file. On input:

Parameter:0 contains the file channel to operate on.

On output:

Parameters:1..4 contains the size of the file.

This call should be used on open files, and takes into account any buffered data which has not yet been written to disk. Consequently, this may return a different size than Function 3,16 "File Stat".

Function 11 : File Set Size

Extends or truncates an opened file to a particular size. On input:

Parameter:0 contains the file channel to operate on;

Parameters:1..4 contains the new size of the file.

Function 12 : File Rename

Renames a file. On input:

Parameters:0,1 points to the length-prefixed string for the old name;

Parameters:2,3 points to the length-prefixed string for the new name.

Files may be renamed across directories.

Function 13 : File Delete

Deletes a file or directory. On input:

Parameters:0,1 points to the length-prefixed filename string.

Deleting a file which is open has undefined behaviour. Directories may

only be deleted if they are empty.

Function 14 : Create Directory

Creates a new directory. On input:

Parameters:0,1 points to the length-prefixed filename string.

Function 15 : Change Directory

Changes the current working directory. On input:

Parameters:0,1 points to the length-prefixed path string.

Function 16 : File Stat

Retrieves information about a file by name. On input:

Parameters:0,1 points to the length-prefixed filename string.

Parameters:0..3 contains the length of the file;

Parameter:4 contains the attributes bit-field of the file.

If the file is open for writing, this may not return the correct size due to buffered data not having been flushed to disk.

File attributes are a bitfield as follows: 0,0,0,Hidden, Read Only, Archive, System, Directory.

Function 17 : Open Directory

Opens a directory for enumeration. On input:

Parameters:0,1 points to the length-prefixed filename string.

Only one directory at a time may be opened. If a directory is already open when this call is made, it is automatically closed. However, an open directory may make it impossible to delete the directory; so closing the directory after use is good practice.

Function 18 : Read Directory

Reads an item from the currently open directory. On input:

Parameters:0,1 points to a length-prefixed buffer for returning the filename.

Parameters:0,1 is unchanged, but the buffer is updated to contain the

length-prefixed filename (without any leading path);

Parameters:2..5 contains the length of the file;

Parameter:6 contains the file attributes, as described by Function 3,16 "File Stat".

If there are no more items to read, this call fails and an error is flagged.

Function 19 : Close Directory

Closes any directory opened previously by Function 3,17 "Open Directory".

Function 20 : Copy File

Copies a file. On input:

Parameters:0,1 points to the length-prefixed old filename;

Parameters:2,3 points to the length-prefixed new filename.

Only single files may be copied, not directories.

Function 21 : Set file attributes

Sets the attributes for a file. On input:

Parameters:0,1 points to the length-prefixed filename;

Parameter:2 is the attribute bitfield. (See Stat File for details.)

The directory bit cannot be changed. Obviously.

Function 22 : Check End of File.

Returns the end of file status of an opened file. On input:

Parameter:0 contains the file channel to operate on.

On output:

Parameter:0 is non-zero if the file is at the end of the file.

This call should be used on open files and may return an error if the file is closed.

Function 23 : Get Current Working Directory.

Copies the current working directory into the String at address Parameters:0,1. which is of maximum length Parameters:2

Function 32 : List Filtered

Prints a filtered file listing of the current directory to the console. On input:

Parameters:0,1 points to the filename search string.

Files will only be shown if the name contains the search string (ie: a substring match).

3.12 Group 4 : Mathematics

Function 0 : Addition

Register1 := Register 1 + Register2

Function 1 : Subtraction

Register1 := Register 1 - Register2

Function 2 : Multiplication

Register1 := Register 1 * Register2

Function 3 : Decimal Division

Register1 := Register 1 / Register2 (floating point)

Function 4 : Integer Division

Register1 := Register 1 / Register2 (integer result)

Function 5 : Integer Modulus

Register1 := Register 1 mod Register2

Function 6 : Compare

Parameter:0 := Register 1 compare Register2 : returns $FF, 0, 1 for less equal and greater.

Note: float comparison is approximate because of rounding.

Function 7 : Power

Register1 := Register 1 to the power of Register2 (floating point result whatever)

Function 8 : Distance (counter-rectangle)

Register1 := Square root of (Register1 * Register1) + (Register2 * Register2)

Function 9 : Angle calculation (arctangent2)

Register1 := arctangent2(Register 1,Register 2) - angle in degrees/radians

Function 16 : Negate

Register1 := -Register 1

Function 17 : Floor

Register1 := floor(Register 1)

Function 18 : Square Root

Register1 := square root(Register 1)

Function 19 : Sine

Register1 := sine(Register 1) angles in degrees/radians

Function 20 : Cosine

Register1 := cosine(Register 1) angles in degrees/radians

Function 21 : Tangent

Register1 := tangent(Register 1) angles in degrees/radians

Function 22 : Arctangent

Register1 := arctangent(Register 1) angles in degrees/radians

Function 23 : Exponent

Register1 := e to the power of Register 1

Function 24 : Logarithm

Register1 := log(Register 1) natural logarithm

Function 25 : Absolute Value

Register1 := absolute value(Register 1)

Function 26 : Sign

Register1 := sign(Register 1), returns -1 0 or 1

Function 27 : Random Decimal

Register1 := random float from 0-1

Function 28 : Random Integer

Register1 := random integer from 0 to (Register 1-1)

Function 32 : Number to Decimal

Helper function for tokeniser, do not use.

Function 33 : String to Number

Convert the length prefixed string at Parameters:4,5 to a constant in Register1.

Function 34 : Number to String

Convert the constant in Register1 to a length prefixed string which is stored at Parameters:4,5

Function 35 : Set Degree/Radian Mode

Sets the use of degrees (the default) when non zero, radians when zero.

3.13 Group 5 : Graphics

Function 1 : Set Defaults

Configure the global graphics system settings.

Not all parameters are relevant for all graphics commands; but all parameters will be set by this command. So mind their values.

Refer to Section "Graphics Settings" for details.

The parameters are And, Or, Fill Flag, Extent, and Flip. Bit 0 of flip sets the horizontal flip, Bit 1 sets the vertical flip.

Function 2 : Draw Line

Draw a line between the screen coordinates specified in Parameters:0,1,Parameters:2,3 (begin X,Y) and Parameters: 4,5,Parameters:6,7 (end X,Y).

Function 3 : Draw Rectangle

Draw a rectangle spanning the screen coordinates specified in Parameters:0,1,Parameters:2,3 (corner X,Y) and Parameters: 4,5,Parameters:6,7 (opposite corner X,Y).

Function 4 : Draw Ellipse

Draw an ellipse spanning the screen coordinates specified in Parameters:0,1,Parameters:2,3 (corner X,Y) and Parameters: 4,5,Parameters:6,7 (opposite corner X,Y).

Function 5 : Draw Pixel

Draw a single pixel at the screen coordinates specified in Parameters:0,1,Parameters:2,3 (X,Y).

Function 6 : Draw Text

Draw the length-prefixed string of text stored at the memory location specified in Parameters:4,5 at the screen character cell specified in Parameters:0,1,Parameters:2,3 (X,Y).

Function 7 : Draw Image

Draw the image with image ID in Parameter:4 at the screen coordinates Parameters:0,1,Parameters:2,3 (X,Y). The extent and flip settings influence this command.

Function 8 : Draw Tilemap

Draw the current tilemap at the screen coordinates specified in Parameters:0,1,Parameters:2,3 (top-left X,Y) and Parameters: 4,5,Parameters:6,7 (bottom-right X,Y) using current graphics settings.

Function 32 : Set Palette

Set the palette colour at the index spcified in Parameter:0 to the values in Parameter:1,Parameter:2,Parameter:3 (RGB).

Function 33 : Read Pixel

Read a single pixel at the screen coordinates specified in Parameters:0,1,Parameters:2,3 (X,Y).

When the routine completes, the result will be in Parameter:0. If sprites are in use, this will be the background only (0..15), if sprites are not in use it may return (0..255)

Function 34 : Reset Palette

Reset the palette to the defaults.

Function 35 : Set Tilemap

Set the current tilemap.

Parameters:0,1 is the memory address of the tilemap, and Parameters:2,3,Parameters:4,5 (X,Y) specifies the offset into the tilemap, in units of pixels, of the top-left pixel of the tile.

Function 36 : Read Sprite Pixel

Read Pixel from the sprite layer at the screen coordinates specified in Parameters:0,1,Parameters:2,3 (X,Y).

When the routine completes, the result will be in Parameter:0.

Refer to Section "Pixel Colors" for details.

Function 37 : Frame Count

Deposit into Parameters:0..3, the number of v-blanks (full screen redraws) which have occurred since power-on. This is updated at the start of each v-blank period.

Function 38 : Get Palette

Get the palette colour at the index spcified in Parameter:0. Values are returned in Parameter:1,Parameter:2,Parameter:3 (RGB).

Function 39 : Write Pixel

Write Pixel index Parameter:4 to the screen coordinate specified in Parameters:0,1,Parameters:2,3 (X,Y).

Function 64 : Set Color

Set Color

Sets the current drawing colour to Parameter:0

Function 65 : Set Solid Flag

Set Solid Flag

Sets the solid flag to Parameter:0, which indicates either solid fill (for shapes) or solid background (for images and fonts)

Function 66 : Set Draw Size

Set Draw Size

Sets the drawing scale for images and fonts to Parameter:0

Function 67 : Set Flip Bits

Set Flip Bits

Sets the flip bits for drawing images. Bit 0 set causes a horizontal flip, bit 1 set causes a vertical flip.

3.14 Group 6 : Sprites

Function 1 : Sprite Reset

Reset the sprite system.

Function 2 : Sprite Set

Set or update the sprite specified in Parameter:0.

The parameters are : Sprite Number, X Low, X High, Y Low, Y High, Image, Flip and Anchor and Flags

Bit 0 of flags specifies 32 bit sprites.

Values that are $80 or $8080 are not updated.

Function 3 : Sprite Hide

Hide the sprite specified in Parameter:0.

Function 4 : Sprite Collision

Parameter:0 is non-zero if the distance is less than or equal to Parameter:2 between the center of the sprite with index specified in Parameter:0 and the center of the sprite with index specified in Parameter:1 .

Function 5 : Sprite Position

Deposit into Parameters:1..4, the screen coordinates of the sprite with the index specified in Parameter:0.

3.15 Group 7 : Controller

Function 1 : Read Default Controller

This reads the status of the base controller into Parameter:0, and is a compatibility API call.

The base controller is the keyboard keys (these are WASD+OPKL or Arrow Keys+ZXCV) or the gamepad controller buttons. Either works.

The 8 bits of the returned byte are the following buttons, most significant first :

Y X B A Down Up Right Left

Function 2 : Read Controller Count

This returns the number of game controllers plugged in to the USB System into Parameter:0. This does not include the keyboard based controller, only physical controller hardware.

Function 3 : Read Controller

This returns a specific controller status. Controller 0 is the keyboard controller, Controllers 1 upwards are those physical USB devices.

This returns a 32 bit value in Parameters:0..3 which currently is compatible with function 1, but allows for expansion.

The 8 bits of the returned byte are the following buttons, most significant first :

Y X B A Down Up Right Left

3.16 Group 8 : Sound

Function 1 : Reset Sound

Reset the sound system. This empties all channel queues and silences all channels immediately.

Function 2 : Reset Channel

Reset the sound channel specified in Parameter:0.

Function 3 : Beep

Play the startup beep immediately.

Function 4 : Queue Sound

Queue a sound. Refer to Section #\ref{sound} "Sound" for details.

The parameters are : Channel, Frequency Low, Frequency High, Duration Low, Duration High, Slide Low, Slide High and Source.

Function 5 : Play Sound

Play the sound effect specified in Parameter:1 on the channel specified in Parameter:0 immediately, clearing the channel queue.

Function 6 : Sound Status

Deposit in Parameter:0 the number of notes outstanding before silence in the queue of the channel specified in Parameter: 0, including the current playing sound, if any.

Function 7 : Queue Sound Extended

Queue a sound. Refer to Section #\ref{sound} "Sound" for details. This is an extension of call 4 to support different waveform types and volumes. The source parameter is no longer used.

The parameters are : Channel, Frequency Low, Frequency High, Duration Low, Duration High, Slide Low, Slide High, Sound Type and Sound Volume. All these

are 16 bit parameters except the sound type and volume, and the channel number.

Function 8 : Get Channel Count

This returns the number of channels in Parameter #0

3.17 Group 9 : Turtle Graphics

Function 1 : Turtle Initialise

Initialise the turtle graphics system.

Parameter:0 is the sprite number to use for the turtle,as the turtle graphics system “adopts” one of the sprites.

The icon is not currently re-definable, and initially the turtle is hidden.

Function 2 : Turtle Turn

Turn the turtle right by Parameter:0,1 degrees. Show if hidden. To turn left, turn by a negative amount.

Function 3 : Turtle Move

Move the turtle forward by Parameter:0,1 degrees, drawing in colour Parameter:2 if Parameter:3 is non-zero.

Function 4 : Turtle Hide

Hide the turtle.

Function 5 : Turtle Home

Move the turtle to the home position (in the center, pointing upward).

Function 6 : Turtle Show

Show the turtle.

3.18 Group 10 : UExt I/O

Function 1 : UExt Initialise

Initialise the UExt I/O system.

This resets the IO system to its default state, where all UEXT pins are I/O pins, inputs and enabled.

Function 2 : Write GPIO

This copies the value Parameter:1 to the output latch for UEXT pin Parameter:0.

This will only display on the output pin if it is enabled, and its direction is set to "Output" direction.

Function 3 : Read GPIO

If the pin is set to "Input" direction, reads the level on pin on UEXT port Parameter:0.

If it is set to "Output" direction, reads the output latch for pin on UEXT port Parameter:0.

If the read is successful, the result will be in Parameter:0.

Function 4 : Set Port Direction

Set the port direction for UEXT Port Parameter:0 to the value in Parameter:1.

This can be $01 (Input), $02 (Output), or $03 (Analogue Input).

Function 5 : Write I2C

Write to I2C Device Parameter:0, Register Parameter:1, value Parameter:2.

No error is flagged if the device is not present.

Function 6 : Read I2C

Read from I2C Device Parameter:0, Register Parameter:1.

If the read is successful, the result will be in Parameter:0.

If the device is not present, this will flag an error.

Use FUNCTION 10,2 first, to check for its presence.

Function 7 : Read Analog

Read the analogue value on UEXT Pin Parameter:0.

This has to be set to analogue type to work.

Returns a value from 0..4095 stored in Parameters:0,1, which represents an input value of 0 to 3.3 volts.

Function 8 : I2C Status

Try to read from I2C Device Parameter:0.

If present, then Parameter:0 will contain a non-zero value.

Function 9 : Read I2C Block

Try to read a block of memory from I2C Device Parameter:0 into memory at Parameters:1,2, length Parameters:3,4.

Function 10 : Write I2C Block

Try to write a block of memory to I2C Device Parameter:0 from memory at Parameters:1,2, length Parameters:3,4.

Function 11 : Read SPI Block

Try to read a block of memory from SPI Device into memory at Parameters:1,2, length Parameters:3,4.

Function 12 : Write SPI Block

Try to write a block of memory to SPI Device from memory at Parameters:1,2, length Parameters:3,4.

Function 13 : Read UART Block

Try to read a block of memory from UART into memory at Parameters:1,2, length Parameters:3,4. This can fail with a timeout.

Function 14 : Write UART Block

Try to write a block of memory to UART from memory

at Parameters:1,2, length Parameters:3,4.

Function 15 : Set UART Speed and Protocol

Set the Baud Rate and Serial Protocol for the UART interface. The baud rate is in Parameters:0..3 and the protocol number is Parameter:4. Currently only 8N1 is supported, this is protocol 0.

Function 16 : Write byte to UART

Write byte Parameter:0 to the UART

Function 17 : Read byte from UART

Read a byte from the UART. It is returned in Parameter:0

Function 18 : Check if Byte Available

See if a byte is available in the UART input buffer. If available Parameter:0 is non zero.

3.19 Group 11 : Mouse

Function 1 : Move display cursor

Positions the display cursor at Parameters:0,1,Parameters:2,3

Function 2 : Set mouse display cursor on/off

Shows or hides the mouse cursor depending on the Parameter:0

Function 3 : Get mouse state

Returns the mouse position (screen pixel, unsigned) in x Parameters:0,1 and y Parameters:2,3, buttonstate in Parameter:4 (button 1 is 0x1, button 2 0x2 etc., set when pressed), scrollwheelstate in Parameter:5 as uint8 which changes according to scrolls.

Function 4 : Test mouse present

Returns non zero if a mouse is plugged in in Parameter:0

Function 5 : Select mouse Cursor

Select a mouse cursor in Parameter:0 ; returns error status if the cursor is not available.

3.20 Group 12 : Blitter

Function 1 : Blitter Busy

Returns a non zero value in Parameter:0 if the blitter/DMA system is currently transferring data, used to check availability and transfer completion.

Function 2 : Simple Blit Copy

Copy Parameters:6,7 bytes of internal memory from Parameter:0:Parameters:1,2 to Parameter:3:Parameters:4,5. Sets error flag if the transfer is not

possible (e.g. illegal write addresses). The upper 8 bits of the address are : 6502 RAM (00) VideoRAM (80,81) Graphics RAM(90)

Function 3 : Complex Blit Copy

Copy a source rectangular area to a destination rectangular area.

It's oriented toward copying graphics data, but can be used as a more general-purpose memory mover.

The source and target areas may be different formats, and the copy will convert the data on the fly.

For example, you can expand 4bpp source graphics (two pixels per byte) into the 1 pixel per byte framebuffer.

However, the blitting is byte-oriented. So the source width is always rounded down to the nearest full byte.

Parameter (0) is the blit action:

0 = copy

1 = copymasked - copy, but only where src is not the transparent value.

2 = solidmasked - set target to constant solid value, but only where src is not the transparent value.

See below for transparent/solid values.

Parameters (1,2) address of the source rectangle data.

Parameters (3,4) address of the target rectangle data.

The source and target rectangle data is laid out in memory as follows:

0-2 24 bit address to copy from/to (address is address:page:0)

3 pad byte (must be zero)

4-5 Stride, in bytes. This is the value to add to the address to get from one line to the next.

Used for both source and target.

For example:

- if blitting to the screen, a stride of screen width (320) would get to the next line.
- a zero source stride would repeat a single line for the whole copy.
- A negative target stride would draw from the bottom upward.

6 data format

0: bytes. Supported for both source and target.

1: pairs of 4-bit values (nibbles). Source only.

2: 8 single-bit values. Source only.

3: high nibble. Target only.

4: low nibble. Target only.

7 A constant to use as the "transparent" value for BLTACT_MASK and BLTACT_SOLID. Source only. Not used in target.

8 A constant to use as the "solid" value for BLTACT_SOLID. Source only. Not used in target.

9 Height. The number of lines to copy.

Source only. Not used in target.

The copy is driven by the source height.

10-11 Width. The number of values to copy for each line.

Source only. Not used in target.

The copy is driven by the source width.

Function 4 : Blit Image

Blits an image from memory onto the screen. The image will be clipped, so it's safe to blit partly (or fully) offscreen-images.

Parameter (0) is the blit action (see function 3, Complex Blit):

Parameters (1,2) address of the source rectangle data.

Parameters (3,4) x pixel coordinate on screen (signed 16 bit)

Parameters (5,6) y pixel coordinate on screen (signed 16 bit)

Parameter (7) destination format, determines how framebuffer will be written:

0: write to whole byte.

1: unsupported

2: unsupported

3: write to high nibble only.

4: write to low nibble only.

NOTE: clipping operates at byte resolution on the source data. So, for example, if you blit a 1-bit image (format 2) to an x-position of -2, then the whole first byte will be skipped leaving 6 empty pixels on the left. Same happens on the right - either the whole source byte is used, or it'll be skipped.

3.21 Group 13 : Editor

Function 1 : Initialise Editor

Initialises the editor

Function 2 : Reenter the Editor

Re-enters the system editor. Returns the function required for call out, the editors sort of 'call backs' - see editor specification.

3.22 Basic Reference

This is a reference for Neo6502's BASIC interpreter.

There are many example programs available which are designed to show and explain the features of the Neo6502 and its hardware and firmware in the examples directory of the release.

For example if.bsc/if.bas so the options available for the 'if' statement, graphics.bsc/graphics.bas show the drawing commands, and joypad.bsc/joypad.bas show how to access the joypad (or the keyboard backup if none is plugged in)

Many of these are helpful for understanding specific API functions, as many BASIC commands are just wrappers for those functions.

3.22.1 Binary Operators

Precedence	Operator	Notes
4	*	
4	/	Forward slash is floating point divide. 22/7 is 3.142857
4	\	Backward slash is integer divide, 22\7 is 3
4	%	Modulus of integer division ignoring signs
4	>>	Logical shift right, highest bit zero
4	<<	Logical shift left
3	+	
3	-	
2	<	Return -1 for true,0 for false
2	<=	Return -1 for true,0 for false
2	>	Return -1 for true,0 for false
2	>=	Return -1 for true,0 for false
2	<>	Return -1 for true,0 for false
2	=	Return -1 for true,0 for false
1	&	Binary AND operator on integers
1	|	Binary OR operator on integers
1	^	Binary XOR operator on integers

3.22.2 Arithmetic functions

Operator	Notes
alloc(n)	Allocate n bytes of memory, return address
analog(n)	Read voltage level on pin n -- returns a value from 0 to 4095
asc(s$)	Return ASCII value of first character or zero for empty string
atan(n)	Arctangent of n in degrees
atan2(y,x)	Arctangent 2 calculation, dy / dx => angle
chr$(n)	Convert ASCII to string
cos(n)	Cosine of n, n Is in degrees.
deek(a)	Read word value at a
eof(f)	Returns non-zero value if at end of file f.
err	Current error number
erl	Current error line number
event(v,r)	event takes an integer variable and a fire rate (r) in 1/100 s, and uses the integer variable to return -1 at that rate. If the value in 'v' is zero, it resets (if you pause say), if the value in v is -1 the timer will not fire -- to unfreeze, set it to zero and it will resynchronise.
exists(file$)	Returns true (-1) if the file exists, false (0) otherwise
exp(n)	e to the power n
false	Return constant 0, improves boolean readability
havemouse()	Return non zero if a mouse is connected.
himem	First byte after end of memory -- the stack is allocated below here, and string memory below that.
inkey$()	Return the key stroke if one is in the keyboard buffer, otherwise returns a n empty string.
idevice(device)	Returns true if i2c device present.
iread(device,register)	Read byte from I2C Device Register
instr(str$,search$)	Returns the first position of search$ in str$, indexed from 1. Returns zero if not found.
int(n)	Whole part of the float value n. Integers are unchanged.
isval(s$)	Converts string to number, returns -1 if okay, 0 if fails.
joycount()	Read the number of attached joypads, not including keyboard emulation of one.
joypad([index],dx,dy)	Reads the current joypad. The return value has bit 0 set if A is pressed, bit 1 set if B is pressed. Values -1,0 or 1 are placed into dx,dy representing movement on the D-Pad. If there is no gamepad plugged in (at the time of writing it doesn't work) the key equivalents are WASDOP and the cursor keys. If [index] is provided it is a specific joypad (from 1,0 is the keyboard), otherwise it is a composite of all of them.
key(n)	Return the state of the given key. The key is the USB HID key scan code.
left$(a$,n)	Left most n characters of a$
len(a$)	Return length of string in characters.
locale a$	Sets the locale to the 2 character country code a$ e.g. locale "de"
log(n)	Natural Logarithm (e.g. ln2) of n.
lower$(a$)	Convert a string to lower case
max(a,b)	Return the largest of a and b (numbers or strings)
mid$(a$,f[,s])	Characters from a$ starting at f (1 indexed), s characters, s is optional and defaults to the rest of the line.

Operator	Notes
min(a,b)	Return the smaller of a and b (numbers or strings)
mos(command)	Like the mos command, but returns an non zero error code if the command caused an error.
mouse(x,y[,scroll])	Reads the mouse. The return value indicates button state (bit 0 left, bit 1 right), and the mouse position and also the scrolling wheel position are updated into the given variables.
not [term]	Returns the logical not of a term.
notes(c)	Return the number of notes outstanding on channel c including the one currently playing -- so will be zero when the channel goes silent.
page	Return the address of the program base (e.g. the variable table)
peek(a)	Read byte value at a
pin(n)	Return value on UEXT pin n if input, output latch value if output.
point(x,y)	Read the screen pixel at coordinates x,y. This is graphics data only.
pow(a,b)	Returns a raised to the power b ; the result is always floating point.
rand(n)	Random integer 0 { x { n (e.g. 0 to n-1)
right$(a$,n)	Rightmost n characters of a$
rnd(n)	Random number 0 { x { 1, ignores n.
sin(n)	Sine of n, n Is in degrees.
spc(n)	Returns a string of n spaces.
spoint(x,y)	Reads the colour index on the sprite layer. 0 is transparency
sqr(n)	Square root of n
str$(n)	Convert n to a string
tab(n)	Advance to screen column n if not past it already.
tan(n)	Tangent of n, n Is in degrees.
true	Return constant -1, improves boolean readability
time()	Return time since power on in 100^th^ of a seconds.
uhasdata()	Return true if there is data in the UART Receive buffer.
upper$(a$)	Convert a string to upper case
val(s$)	Convert string to number. Error if bad number.
vblanks()	Return the number of vblanks since power on. This is updated at the start of the vblank period.

3.22.3 BASIC Commands (General)

Command	Notes
' {string}	Comment. This is a string for syntactic consistency. The tokeniser will process a line that doesn't have speech marks as this is not common, so you can type in ' hello world and it will be represented as ' "hello world" in code.
assert {expr}[, {msg}]	Error generated if {expr} is zero, with optional message.
call {name} (p1,p2,p3)	Call named procedure with optional parameters.
cat [{pattern}]	Show contents of current directory, can take an optional string which only displays filenames containing those characters, so cat "ac" only displays files with the sequence ac in them.
clear [{address}]	Clear out stack, strings, reset all variables. If an address is provided then memory above that will not be touched by BASIC. Note because this resets the stack, it cannot be done in a loop, subroutine or procedure -- they will be forgotten. Also clears the sprites and the sprite layer.
close [handle]	Close a file by handle. If the handle is not provided, close all files.
cls	Clear the graphics screen to current background colour. This does not clear sprites.
cursor {x},{y}	Set the text cursor position
data {const},....	DATA statement. For syntactic consistency, strings must be enclosed in quote marks e.g. data
defchr ch,....	Define UDG ch (192-255) as a 6x7 font -- should be followed by 7 values from 0-63 representing the
delete	Delete a line or range of lines
dim {array}(n,[m]), $...	Dimension a one or two dimension string or number array, up to 255
do ... exit ... loop	General loop you can break out of at any point.
doke {addr},{data}	Write word to address
edit	Basic Screen Editor
end	End Program
fkey	Lists the defined function keys
fkey {key},{string}	Define the behaviour of F1..F10 -- the characters in the string
for {var} = {start} to/downto	For loop. Note this is non standard,Limitations are : the index must be an integer. Step can only be 1 (to) or -1 (downto). Next does not specify an index and cannot be used to terminate loops using the 'wrong' index.
gload {filename}	Load filename into graphics memory.
gosub {expr}	Call subroutine at line number. For porting only. See goto.
goto {expr}	Transfer execution to line number. For porting only. Use in general coding is a capital offence. If I write RENUMBER it
if {expr} then	Standard BASIC if, executes command or line number. (IF .. GOTO doesn't work, use IF .. THEN nn)
if {expr}: .. else .. endif	Extended multiline if, without THEN. The else clause is optional.
ink fgr[,bgr]	Set the ink foreground and optionally background for the console.
input {stuff}	Input has an identical syntax and behaviour to Print except that variables are entered via the keyboard rather than printed.
input #{channel}, {var},{var}	Reads a sequence of variables from the open file.

Command	Notes
input line #{channel}.{var}.{var}	Reads text from an ASCII file. This is processed from the source, primarily due to Windows' usage of CR/LF. So this at current, by default, ignores all characters before space, except for LF (10) which marks the end of line, and TAB (9) which is converted into a space. All variables are strings
ireceive {d},{a},{s}	Send or receive bytes starting at a, count s to or from device d.
itransmit {d},{a},{s}	
isend {device},{data}	Send data to i2c {device} ; this is comma seperated data, numbers or strings. If a semicolon is used as a seperator e.g. 4137; then the constant is sent as a 16 bit value.
iwrite {dev},{reg},{b}	Write byte to I2C Device Register
let {var} = {expr}	Assignment statement. The LET is optional.
library	Librarise / Unlibrarise code.
list [{from}][,][{to}]	List program to display by line number or procedure name.
list {procedure}()	
load "file"[,{address}]	Load file to BASIC space or given address.
local {var},{var}	Local variables, use after PROC, restored at ENDPROC variables can
mon	Enter the machine code monitor
mos {command}	Execute MOS command.
mouse cursor {n}	Select mouse cursor {n} [0 is the default hand pointer]
mouse show	hide
mouse TO {x},{y}	Position mouse cursor
new	Erase Program
next {variable}	Ends for loop. The variable parameter is optional. You cannot unwind nested FOR/NEXTs , next must operate in order.
old	Undoes a new. This can fail depending on what has been done since the 'new'.
on error {code}	Install an error handler that is called when an error occurs. Effectively this is doing a GOTO that code, so recovery is dependent on what you actually
open input|output [channel],[file]	Open a file for input or output on the given channel, using the given file name. Output erases the current file. This gives an error if the file does not exist ; rather than trap this error it is recommended to use the exists() function if you think the file may not be present.
palette c,r,g,b	Set colour c to r,g,b values -- these are all 0-255 however it is actually 3:2:3 colour, so they will be approximations.
palette clear	Reset palette to default
pin {pin},{value}	Set UEXT {pin} to given value.
pin {pin} INPUT	output
poke {addr},{data}	Write byte to address
print {stuff}	Print strings and numbers, standard format - , is used for
print #{channel},{expr},{expr}	Writes a sequence of expressions to the open file.
print line #{channel}.{var}.{var}	Prints a line to an output channel as an ASCII file, in LF format (e.g. lines are seperated by character code 10). This can be mixed with the above format *but* the sequence has to be the same ;

Command	Notes
	you ann't write a string using print line and read it back with input and vice versa. All variables must be strings.
proc {name>([ref] p1,p2,...) .. endproc	Delimits procedures, optional parameters, must match call. Parameters can be defined as reference parameters and will return values. Parameters cannot be arrays.
read {var},...	Read variables from data statements. Types must match those in data statements.
renumber [{start}]	Renumber the program from start, or from 1000 by default. This does *not* handle GOTO and GOSUB. Use those, you are on your own.
repeat .. until {expr}	Execute code until {expr} is true
restore	Restore data pointer to program start
restore {line}	Restore data pointer to line number
return	Return from subroutine called with gosub.
run	Run Program
run "{program}"	Load & Run program.
save "file"[,{adr},{sz}]	Save BASIC program or memory from {adr} length {sz}
sreceive {a},{s}	Send or receive bytes starting at a, count s to SPI device
stransmit {a},{s}	
ssend {data}	Send data to SPI device ; this is comma seperated data, numbers or strings. If a semicolon is used as a seperator e.g. 4137; then the constant is sent as a 16 bit value.
stop	Halt program with error
sys {address}	Call 65C02 machine code at given address. Passes contents of variables A,X,Y in those registers.
tilemap addr,x,y	Define a tilemap. The tilemap data format is in the API. The tilemap is stored in memory at addr, and the offset into the
uconfig {baud}[,{prt}]	Set the baud rate and protocol for the UART. Currently only 8N1 is supported.
ureceive {d},{a},{s}	Send or receive bytes to/from the UART starting at a, count s
utransmit {d},{a},{s}	
usend {device},{data}	Send data to UART ; this is comma seperated data, numbers or strings. If a semicolon is used
wait {cs}	Waits for {cs} hundredths of a second
while {expr} .. wend	Repeat code while expression is true
who	Display contributors list.

3.22.4 The Inline Assembler

The inline assembler works in a very similar way to that of the BBC Micro, except that it does not use the square brackets [and] to delimit assembler code. Assembler code is in normal BASIC programs.

A simple example shown below (in the samples directory). It prints a row of 10 asterisks.

Most standard 65C02 syntax is supported, except currently you cannot use lsr a ; it has to be just lsr (and similarly for rol, asl, ror,inc and dec).

You can also pass A X Y as variables. So you could delete line 150 and run it with X = 12: sys start which would print 12 asterisks.

Line	Code	Notes
100	mem = alloc(32)	Allocate 32 bytes of memory to store the program code.
110	for i = 0 to 1	We pass through the code twice because of forward referenced labels. This actually doesn't apply here.
120	p = mem	P is the code pointer -- it is like $* = {xx} - it means put the code here
130	o = i * 3	Bit 0 is the pass (0 or 1) Bit 1 should display the code generated on pass 2 only, this is stored in 'O' for options.
140	.start	Superfluous -- creates a label 'start' -- which contains the address here
150	ldx #10	Use X to count the starts
160	.loop1	Loop position. We can't use loop because it's a keyword
170	lda #42	ASCII code for asterisk
180	jsr $fff1	Monitor instruction to print a character
190	dex	Classic 6502 loop
200	bne loop1	
210	rts	Return to caller
220	next	Do it twice and complete both passes
230	sys mem	BASIC instruction to 'call 6502 code'. Could do sys start here.

[ ] Operator

The [] operator is used like an array, but it is actually a syntactic equivalent of deek and doke, e.g. reading and writing 16 bytes. mem[x] means the 16 bit value in mem + x $* 2, so if mem = 813 then mem[2] = -1 writes a 16 bit word to 817 and 818, and print mem[2] reads it. The index can only be from 0..127

The purpose of this is to provide a clean readable interface to data in 65C02 and other programs running under assembly language ; often accessing elements in the 'array' as a structure.

Zero Page Usage

Neo6502 is a clean machine, rather like the Sharp machines in the 1980s. When BASIC is not running it has no effect on anything, nor does the firmware. It is not like a Commodore 64 (for example) where changing some zero page locations can cause crashes.

However, BASIC does make use of zero page. At the time of writing this is memory locations $10-$41.

These can however be used in machine code programs called via SYS. Only 4 bytes of that usage is system critical (the line pointer and the stack pointer), those are saved on the stack by SYS, so even if you overwrite them it does not matter.

However, you can't use this range to store intermediate values *between* sys calls. It is advised that you work usage backwards from $FF (as BASIC is developed forwards from $10). It is very unlikely that these will meet in the middle.

$00 and $01 are used on BASIC boot (and maybe other languages later) but this should not affect anything.

3.22.5 Basic Commands (Graphics)

The graphics commands are MOVE, PLOT (draws a pixel), LINE (draws a line) RECT (draws a rectangle) ELLIPSE (draws a circle or ellipse) IMAGE (draws a sprite or tile), TILEDRAW (draws a tilemap) and TEXT (draws text)

The keywords are followed by a sequence of modifiers and commands which do various things as listed below

Keyword	Notes
from x,y	Sets the origin position, can be repeated and optional.
to x,y	Draw the element at x,y or between the current position and x,y depending on the command. So you could have **text "Hello" to 10,10** or **rect 0,0 to 100,50**
by x,y	Same as to but x and y are an offset from the current position
x,y	Set the current position without doing the action
ink c	Draw in solid colour c
ink a,x	Draw by anding the colour with a, and xoring it with x.
solid	Fill in rectangles and ellipses. For images and text, forces black background.
frame	Just draw the outline of rectangles and ellipses
dim n	Set the scaling to n (for TEXT, IMAGE, TILEMAP only), so **text "Hello" dim 2 to 10,10 to 10,100** will draw it twice double size. Tiles can only be 1 or 2 (when 2, tiles are drawn double size giving a 32x32 tile map)

These can be arbitrarily chained together so you can do (say) LINE 0,0 TO 100,10 TO 120,120 TO 0,0 to draw an outline triangle. You can also switch drawing type in mid command, though I probably wouldn't recommend it for clarity.

State is remember until you clear the screen so if you do INK 2 in a graphics command things will be done in colour 2 (green) until finished.

TEXT is followed by one parameter, which is the text to be printed, these too can be repeated

```
TEXT "Hello" TO 10,10 TEXT "Goodbye" DIM 2 TO 100,10
```

IMAGE is followed by two parameters, one specifies the image, the second the 'flip'. These can be repeated as for TEXT.

The image parameter is 0-127 for the first 128 tiles, 128-191 for the first 64 16x16 sprites and 192-255 for the first 64 32x32 sprites. The flip parameter, which is optional, is 0 (no flip) 1 (horizontal flip) 2 (vertical flip) 3 (both).

An example would be

```
image 4 dim 2 to 10,10 image 192,3 dim 1 to 200,10
```

Note that images are $*not$* sprites or tiles, they use the image to draw on the screen in the same way that LINE etc. do.

3.22.6 Sprite Commands

Sprite commands closely resemble the graphics commands.

They begin with SPRITE {n} which sets the working sprite. Options include IMAGE {n} which sets the image, TO {x},{y} which sets the position, FLIP {n} which sets the flip to a number (bit 0 is horizontal flip, bit 1 is vertical flip), ANCHOR {n} which sets the anchor point and BY {x},{y} which sets the position by offset.

With respect to the latter, this is the position from the TO and is used to do attached sprites e.g. you might write.

```
SPRITE 1 IMAGE 2 TO 200,200 SPRITE 2 IMAGE 3 BY 10,10
```

Which will draw Sprite 1 and 200,200 and sprite 2 offset at 210,210. It does not offset a sprite from its current position.

As with Graphics these are not all required. It only changes what you specify not all elements are required each time *SPRITE 1 IMAGE 3* is fine.

SPRITE can also take the single command CLEAR ; this resets all sprites and removes them from the display

Sprite 127 is used for the turtle sprite, so if the turtle is turned on, then it will adjust its graphic, size to reflect the turtle position. If turtle graphics are not used, it can be used like any other.

Implementation notes

Up to 128 sprites are supported. However, sprite drawing is done by the Pico and is not hardware, so more sprites means the system will run slower.

Additionally, the sprites are currently done with XOR drawing, which causes effects when they overlap. This should not be relied on (it may be replaced by a clear/invalidate system at some point), but the actual implementation should not change.

This is an initial sprite implementation and is quite limited.

(The plan is to add a feature like the animation languages on STOS and AMOS which effectively run a background script on a sprite)

3.22.7 Sprite Support

=spritex(n) =spritey(n)

These return the x and y coordinates of the sprites draw position (currently the centre) respectively.

= hit(sprite#1,sprite#2,distance)

The hit function is designed to do sprite collision. It returns true if the pixel distance between the centre of sprite 1 and the centre of sprite 2 is less than or equal to the distance.

So if you wanted to move a sprite until it collided with another sprite, assuming both are 32x32, the collision distance would be 32 (the distance from the centre to the edge of both sprites added together), so you could write something like :

```
x = 0
```

```
repeat
```

```
x = x + 1: sprite 1 to x,40
```

```
until hit(1,2,32)
```

In my experience of this the distance needs to be checked experimentally, as it affects the 'feel' of the game ; sometimes you want near exact collision, sometimes it's about getting the correct feel. It also depends on the shape and size of the sprites, and how they move.

I think it's better than a simple box collision test, and more practical than a pixel based collision test which is very processor heavy.

3.22.8 Sound Commands

The Neo6502 has four sound channels, 0-3 which can generate a square wave or white noise sounds.

The main sound command is called "sound" and has the following forms.

Sound clear

Resets the entire sound system, silences all channels, empties all queues

Sound {channel} clear

Resets a single channel ; silences it, and empties its queue

Sound {channel},{frequency},{time}[,{slide}]

Queues a note on the given channel of the given frequency (in Hz) and time (in centiseconds). These will be played in the background as other notes finish so you can 'queue up' an entire phrase and let it play by itself. The slide value adds that much to the frequency every centisecond allowing some additional effects (note, done in 50Hz ticks)

A mixture of the two syntaxes SOUND 0 CLEAR 440,200 is now supported.

To use the white noise feature use the keyword "noise" instead of sound.

Sfx

Sfx plays sound effects. Sound effects are played immediately as they are usually in response to an event.

It's format is ***sfx**** {channel},{effect***} .

3.22.9 Screen Editor

The ***edit*** command starts the screen editor. This currently supports left, right, home, end, backspace and delete on the current line, enter, up, down, page up and page down to change lines.

Esc exits the editor, and Ctrl+P and Ctrl+Q insert and delete a whole line.

Note the editor is slightly eccentric ; it is not a text editor, what it is doing is editing the underlying program on the fly -- much the same as if you were typing lines in.

The editor uses line numbers, so is *not* compatible with their use in programs. Any program will be renumbered from 1 upwards in steps of 1 (except library routines).

You shouldn't be using line numbers anyway !

3.22.10 Libraries

Libraries are part of the BASIC program, placed at the start. However, their line numbers are all set to zero. (So you cannot use GOTO or GOSUB in libraries, but you should only use them for porting old code).

NEW will not remove them, LIST does not show them (except LIST 0), You cannot edit them using the line editors.

However RUN does not skip them. This is so you can have initialisation code e.g.

{do initialisation code}

```
if false

proc myhw.dosomething(p1) ....

proc myhw.panic()

endif
```

To support this, there is a LIBRARY command. LIBRARY on its own undoes the library functionality. It renumbers the whole program from the start, starting from line number 1000.

Otherwise LIBRARY works like LIST. You can do LIBRARY {line} or LIBRARY {from},{to} and similar. Instead of listing this code, it causes them to "vanish" by setting their line numbers to zero.

They are also supported in the makebasic script. Adding the command library makes all code so far library code.

e.g.

python makebasic.zip mylib.bsc library mainprogram.bsc

3.22.11 Turtle Graphics

The Neo6502 has a built in turtle graphics system. This uses sprite $7F as the turtle, which it will take over, so it cannot be used for other purposes.

The following commands are supported.

Command	Purpose
forward {n}	Move turtle forward n (pixel distance)
left {n}	Rotate turtle at current position
right {n}	
penup	Does not draw as the turtle moves
pendown	Draw as the turtle moves in the current colour
pendown {n}	Draw as the turtle moves in colour {n}
turtle home	Reset turtle to home position, facing up the screen.
turtle hide	Hide the turtle
turtle show	Show the turtle
turtle fast	The turtle is deliberately slowed to give it an animated feel so you can see the drawing, this is because it's primary purpose is educational. This makes it go full speed.

There is an example in the crossdev folder which gives some idea on how to get started.

3.23 File Formats

There are some standard file extensions - .bsc for Basic source (e.g. text file) .bas for Basic tokenised, .neo for a runnable file, but these are not mandatory

3.23.1 NEO Load file format

There is an extended file format which allows the loading of multiple files and optional execution. This is as follows

Offset	Contents	Notes
0	$03	Not a valid 65C02 opcode, nor can it be the first byte of a program.
1	$4E	ASCII 'N'
2	$45	ASCII 'E'
3	$4F	ASCII 'O'
4,5	$00,$00	Minimum major/minor version required to work.
6,7	$FF,$FF	Execute address. To autorun a BASIC program set to $806
8	Control	Control bits, currently only bit 7 is used, which indicates another block follows this one
9,10	Load	Load address (16 bits) $FFFF loads into graphic object memory, $FFFD loads to the BASIC workspace.
11,12	Size	Size to load in bytes.
13...	Comment	ASCIIZ string which is a comment, filename, whatever
....	Data	The data itself

The block then repeats from 'Control' as many times as required.

The Python application 'exec.zip' both constructs executable files, or displays them. This has the same execution format as the emulator, as listed below.

python exec.zip -d{file} dumps a file

python exec.zip {command list} -o{outputfile} builds a file

- {file}@page Loads BASIC program
- {file}@ffff Loads Graphics Object file
- {file}@{hex address} Loads arbitrary file
- run@{hex address} Sets the executable address
- exec runs BASIC program

for example, you can build a frogger executable with:

python exec.zip {frogger.bas@page} {frogger.gfx@ffff} exec -ofrogger.neo

Loading a file can be done by calling the kernel function LoadExtended (which does the autorun for you) or using the normal messaging system.

If you handle it yourself bear in mind that on return, it is always possible that the code you write to call the execution routine may already have been overwritten by the loaded file.

3.24 Graphics

3.24.1 Pixel Colours

Pixel	Colour
0	Black/Transparent
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White
8	Black
9	Dark Grey
10	Dark Green
11	Orange
12	Dark Orange
13	Brown
14	Pink
15	Light Grey

3.24.2 Tile Maps

A tile map occupies an area of user memory in 65C02. It is comprised of three meta-data bytes, followed by one byte for each tile, which is it's tile number in the graphic file (refer to the following section).

F0-FF are special reserved tile numbers, F0 is a transparent tile; and F1-FF are a solid tile in the current palette colour. The format is very simple.

Tile Maps Format

Offset	Data	Notes
0	1	Graphics Data Format ID
1	Width	Width of tile-map (number of tiles)
2	Height	Height of tile-map (number of tiles)
3..	Raw	Tiles graphics data (width * height bytes)

3.24.3 Graphic Data

The graphic data for a game is stored in what is named by default "graphics.gfx". This contains up to 256 graphics objects, in one of three types. One can have multiple graphics files.

Each has 15 colours (for sprites, one is allocated to transparency) which are the same as the standard palette.

16x16 tiles (0-127, $00-$7F)

These are 128 16x16 pixel solid tiles which can be horizontally flipped

16x16 sprites (128-191, $80-$BF)

These are 64 16x16 sprites which can be horizontally and/or vertically flipped

32x32 sprites (192-255, $C0-$FF)

These are 64 32x32 sprites which can be horizontally and/or vertically flipped

These are created using two scripts, which are written in Python and require the installation of the Python Imaging Library, also known as PIL or Pillow.

Empty graphics files

The script "createblanks.zip" creates three files, tile_16.png, sprite_16.png and sprite_32.png which are used for the three types of graphic.

The sprite and tile files all look very similar. The palette is shown at the top (in later versions this will be configurable at this point), and some sample sprites are shown. Each box represents a 16x16 sprite. 32X32 sprite looks the same except the boxes are twice the size and there are half as many per row.

Tiles are almost identical ; in this the background is black. The solid magenta (RGB 255,0,255) is used for transparency, this colour is not in the palette.

Running createblanks.zip creates these three empty files. To protect against accidents it will not overwrite currently existing files, so if you want to start again then you have to delete the current ones.

Compiling graphics files

There is a second script "makeimg.zip". This converts these three files into a file "graphics.gfx" which contains all the graphic data.

This can be loaded into graphics image memory using the gload command, and the address 65535 e,g, ***gload "graphics.gfx"*** or the API equivalent

There is an example of this process in the repository under basic/images which is used to create graphic for the sprite demonstation program

3.25 Memory Map

Note *all this is actually RAM* and functions as it, except for the command area ; writing non zero values into the Command byte may affect other locations in that 16 byte area or elsewhere (e.g. reading a file in).

This block is however moveable.

It is also possible that the top of free memory will move down from $FC00.

Addresses	Contents
0000-FBFF	Free memory. Not used for anything.
FC00-FEFF	Kernel Image. Contains system functions and WozMon, which it currently boots into. This is RAM like everything else.
FFF0-FF0F	Command, Error, Parameters, Information space. This can be moved to accommodate other systems.
FF10-FFF9	Vectors to Kernel routines, not actually mandatory either.
FFFA-FFFF	65C02 Vectors for NMI, IRQ and Reset. This one is mandatory.

3.26 ---

4. Software for the Neo6502

[Dungeon Crawler by Chris Garrett]

A classic dungeon crawler with ASCII graphics.

[Eliza by Erland Nagel]

An implementation of the 1960s faux psychiatrist program.

[Kulki by Wojciech Bocianski]

Kulki is a puzzle game where you have to place marbles in a row

[Mapper by Giovanni Pozzobon]

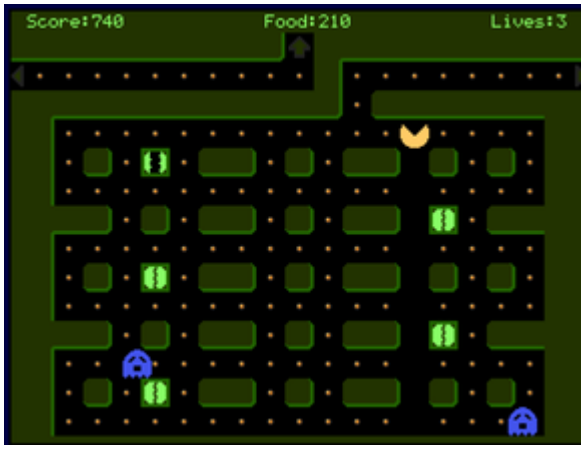
A tilemap editor which can output files in Neo6502's tile format.

Neo 6502 BASIC Demo Games repository (Squash, Space Invaders, Galaxians, Frogger, Asteroids)

Some versions of original classics written to test the capabilities of the machine. These are all in BASIC, except for Asteroids which is partially assembler.

PacMad by Wojciech Bocianski

A scrolling Pacman game, in Pascal.



Power Bricks game by Wojciech Bocianski

A 1024 puzzle game.

Robotron 2084 by Paul Robson

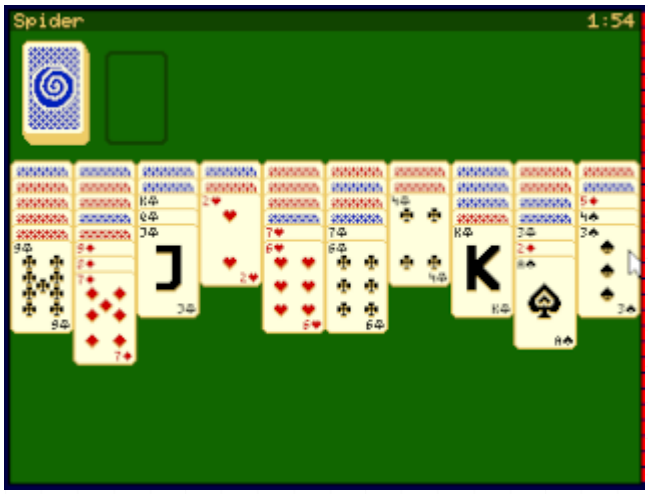
A version of the Williams Arcade classic.

Sokoban by zbyti

The classic puzzle game

Solitaire Suite by Wojciech Bocianksi

A suite of Solitaire games



Swimo by Wojciech Bocianski

A cellular automata type simulator.

TaliFORTH repository by Sam Colwell

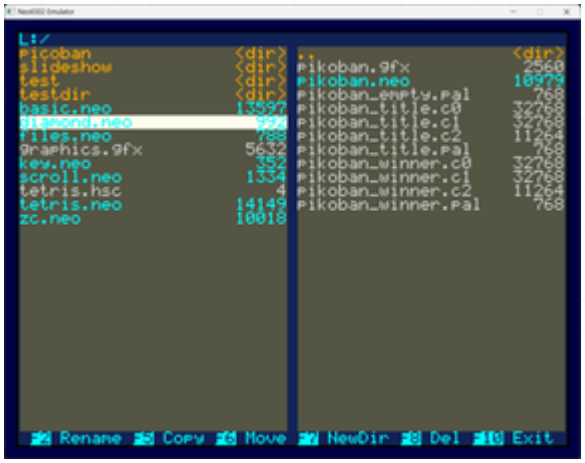
65C02 specific FORTH system.

Tetris by Wojciech Bocianski

Tetris :)

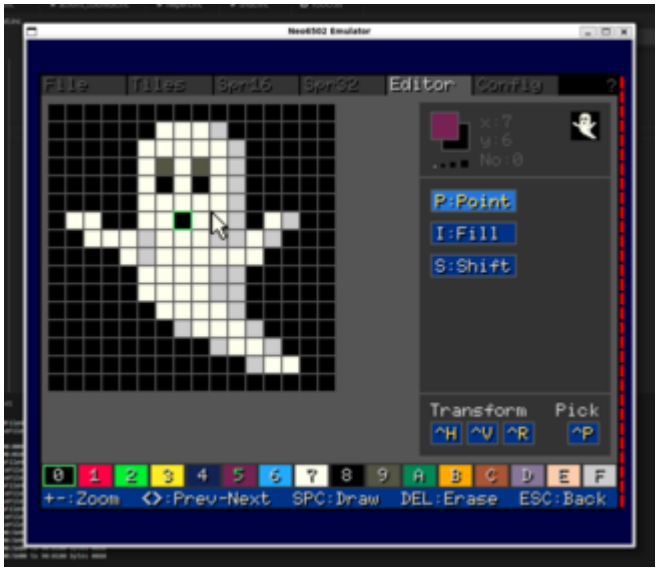
[Zion Commander File Manager by Wojciech Bocianski]

A file management by mouse click program.



[Spoon - neo6502 sprite editor by Wojciech Bocianski]

A sprite editor for use with Neo6502 games.



5. Programming

5.1 Using Assembler

Currently the best start point for assembler are the examples in the release directory.

5.1.1 ---

5.2 Using Mad Pascal

Mad-Pascal is a highly optimized and continuously developed cross-compiler for the Pascal language for the 6502 family of processors. Initially created for the Atari platform, it has since been adapted for several other target platforms, including neo6502.

Compilation to machine code is a two-step process. In the first step, the code is compiled to assembler source code compatible with the MADS assembler, and then the code is assembled into an executable machine code file.

The following guide will help you set up a complete development environment, tailored for quick and convenient coding on the neo6502 on your PC. We will also write our first program together, compile it, and run it in an emulator.

5.2.1 Installing the Emulator

The first thing we will need is the latest version of the neo6502 emulator so that we can test our software on a development machine. Download it from the project's official GitHub: https://github.com/paulscottrobson/neo6502-firmware/releases

All we need for now is the emulator executable. It is located in the root directory of the downloaded archive and is simply named `neo` (`neo.exe` for Windows). Let's place it in an easily accessible directory. For the purposes of our guide, let it be the following directory:

Windows	Linux
`d:/neo6502/neo.exe`	`~/neo6502/neo`

Note that the downloaded archive also contains a `documents` directory, which includes a detailed description of the API functions provided by the firmware of our device. It will certainly come in handy during our adventures with programming on the neo6502, so it's worth keeping it nearby.

5.2.2 Downloading and Compiling the Assembler and Mad-Pascal Compiler

I recommend always using the latest version of the MADS assembler and Mad-Pascal compiler, which we will compile from the sources in a moment. For this purpose, we will need the git tool and the free FreePascal compiler. Both tools should be downloaded from the above links and installed.

Now, using git, we will download the latest versions of the assembler and compiler and then compile them.

Windows

Open a console (cmd), navigate to the directory where you want to install the compilers (in our example `D:`), and then execute the following commands:

```
git clone https://github.com/tebe6502/Mad-Assembler.git
cd Mad-Assembler
fpc -Mdelphi -vh -O3 mads.pas
cd ..
git clone https://github.com/tebe6502/Mad-Pascal.git
cd Mad-Pascal/src
fpc -Mdelphi -vh -O3 mp.pas
copy mp.exe ..
```

Linux

In the directory where you want to install the compilers (in our example in the home directory `~`), enter the following console commands:

```
git clone https://github.com/tebe6502/Mad-Assembler.git
cd Mad-Assembler
fpc -Mdelphi -vh -O3 mads.pas
cd ..
git clone https://github.com/tebe6502/Mad-Pascal.git
cd Mad-Pascal/src
fpc -Mdelphi -vh -O3 mp.pas
cp mp ..
```

Be careful because the **-O3** parameter is an uppercase "o", not a zero.

5.2.3 Writing Our First Program

Now let's write our first Pascal program and save it in the project directory as `hello.pas`:

```
program hello;
begin
  Writeln('Hello neo6502!');
end.
```

Let's try to compile it manually:

Windows	Linux
`D:\Mad-Pascal\mp.exe hello.pas -target:neo -code:5000`	`~\Mad-Pascal\mp hello.pas -target:neo -code:5000`

The `-code:5000` option will cause our program code to be compiled at address $5000 (20480). This is an example value; you can compile the code to any location, just remember not to overwrite areas used by the firmware (you can find the memory map in the emulator archive in the `documents\basic.pdf` directory). As a result of the compilation, an intermediate assembler file `hello.a65` will be created in the project directory. In the next step, we will assemble this program into machine code, i.e., the target executable file.

Windows	Linux
`D:\Mad-Assembler\mads.exe hello.a65 -x -i:D:\Mad-Pascal\base -o:hello.bin`	`~\Mad-Assembler\mads hello.a65 -x -i:~\Mad-Pascal\base -o:hello.bin`

The **-x** option causes potentially unused procedures to be removed during the assembly process, which is beneficial for us as the resulting file will be smaller. The path after the **-i** parameter indicates the directory with the base Mad-Pascal libraries, and the **-o** parameter allows you to specify the name of the output file, which should appear in the project directory after the assembly.

Let's try to run it in the emulator.

Windows	Linux
`D:\neo6502\neo.exe hello.bin@5000 run@5000`	`~\neo6502\neo hello.bin@5000 run@5000`

Hooray! We have successfully written our first program in Mad-Pascal for neo6502! Notice that in the emulator call, we need to specify the compilation and run address consistent with the one provided during the initial compilation.

Entering all these commands each time would be terribly inconvenient and time-consuming, so we will automate this process for our convenience and save time.

5.2.4 Preparing Batch Files for Source Compilation

In the directory of our first project, let's prepare a batch file that will do the hard work for us and carry out the process of compiling and running any Pascal file.

Windows

Let's create a batch file `build.bat`

```
@setlocal
@SET MPPATH=D:\Mad-Pascal
@SET MADSPATH=D:\Mad-Assembler
@SET NEOPATH=D:\neo6502
@SET ORG=5000
@SET NAME=%1

%MPPATH%\mp.exe %NAME%.pas -target:neo -code:%ORG%
@if %ERRORLEVEL% == 0 %MADSPATH%\mads.exe %NAME%.a65 -x -i:%MPPATH%\base -o:%NAME%.bin
@if %ERRORLEVEL% == 0 %NEOPATH%\neo.exe %NAME%.bin@%ORG% run@%ORG%
```

Linux

Let's create a batch file `build.sh`

```
#!/usr/bin/bash
MPPATH=~/Mad-Pascal
MADSPATH=~/Mad-Assembler
NEOPATH=~/neo6502
ORG=5000
NAME=${1}

$MPPATH/mp ${NAME}.pas -target:neo -code:${ORG} && \
$MADSPATH/mads ${NAME}.a65 -x -i:${MPPATH}/base -o:${NAME}.bin && \
$NEOPATH/neo ${NAME}.bin@${ORG} run@${ORG}
```

In the Linux system, the `build.sh` file must have execution rights:

```
chmod +x build.sh
```

Let's try to compile and run our first program using the batch file.

Windows	Linux
`build.bat hello`	`./build.sh hello`

And if everything went well, you should see the compilation process in the console window, and then the emulator window with our first program running. If something went wrong, make sure you have the correct paths in the batch file.

Of course, you can modify and customize this batch file as you see fit, or write it completely differently. This is just an example that I hope will encourage you to experiment with Mad-Pascal.

5.2.5 Libraries for neo6502

The bare "naked" Mad-Pascal will allow us to write simple programs in text mode, but to fully exploit its capabilities on the neo6502, several additional libraries have been created. They help in communication with the device's firmware and provide full support for color graphics, sprites, tile maps, controllers, and all functions provided by the system's API.

These libraries are available to you without any additional steps as they are already part of the Mad-Pascal compiler distribution.

Library Name	Description
NEO6502	Main API function library for Neo6502
NEO6502KEYS	Library for reading keyboard state using HID codes.
NEO6502MATH	Library for accelerating mathematical calculations and some other operations
NEO6502UEXT	Input/output interface function library for UEXT

Link to full documentation: https://bocianu.gitlab.io/neo-mplibs/

To use these libraries, simply add their names in the `uses` clause.

```
program api;
uses neo6502, crt;
begin
  ClrScr;
  NeoBeep;
  NeoCredits;
  repeat until KeyPressed;
end.
```

5.2.6 Sample Programs

More sample programs utilizing specific neo6502 functions can be found in the Mad-Pascal directory in the `Mad-Pascal\samples\neo6502\` subfolder. These are good examples to start with and to begin your adventure with this compiler.

And if you'd like to look at the source code of larger projects in Mad-Pascal, check out bocianu's GitLab, who has written several games and tools for the neo6502.

5.2.7 ---

5.3 Using CC65

There are two references on using CC65

Pete Gollan wrote the following, which has a sample project that can be used as a starter

https://github.com/PeteGollan/Neo6502-programs/tree/main/HelloNeo6502-CC65

Andy McCall wrote a walkthrough and example code on using a combination of LLVM-MOS and CC65 here, which is Linux / Mac centric but the basic ideas could be adapted for Windows.

https://github.com/andymccall/neo6502-development

There are some C examples in the release, but they are considerably more primitive.

5.3.1 ---

5.4 Using LLVM-Mos

With the new version of llvm-mos, which natively supports neo6502, you can compile your C source code (even written for the CC65 compiler) without modifying the code. Pay attention to the warnings; llvm-mos is much stricter than CC65.

Download here

Compile the source following the instruction inside the page

You will find useful files in the directory:

```
$(LLVM_MOS_SDK)/bin/mos-neo6502-clang
```

If you're starting from scratch, you can use the high-level functions for writing to the API mailbox. See the directory:

```
$(LLVM_MOS_SDK)/mos-platform/neo6502/include
```

Here's an example of the command to compile the code:

```
$(LLVM_MOS_SDK)/bin/mos-neo6502-clang  main.c -o main.neo
```

If you use the -Os option, which allows you to have optimized code, remember that the variables with which you call the library should be of type "volatile." Don't make the same mistake as I did. If you use the LLVM_MOS high-level functions don't worry about this. And here's an example of the command to compile the optimized code:

```
$(LLVM_MOS_SDK)/bin/mos-neo6502-clang  -Os main.c -o main.neo
```

If you're using VSCode to write the code, add the directory $(LLVM_MOS_SDK)/mos-platform/neo6502/include to the IncludePath variable.

In the directory, you can find an example of using the API mailbox functions:

```
$(LLVM_MOS_SDK)/examples/neo6502
```

6. Getting Online

You can now get online, with the help of Olimex's ESP8266 based Wifi-board. This costs (at the time of writing) under 4 Euros.

At present we have a program repository, but now we have a solid Wifi connection we should see growth in this in the future.

The 'Getting Started' guide is here

The board can be acquired from here and other Olimex suppliers

How to access the Prophet Server here

Thanks to Wojciech Bocianski for his awesome work on this.

6.1 ---

7. Our Wiki.

Welcome to the Neo6502 wiki. Useful places you may want to go

- Emulator usage
- Cross Development
- Building Firmware
- Building the Emulator
- Autostarting programs
- Serial Interface
- Mos Interface
- Keyboard Locales
- Using Assembler
- Using LLVM (C Programming) for MOS
- Using CC65 (C Programming) for MOS
- Mad-Pascal-for-Neo6502
- Olimex Home Page
- Neo6502 product page
- Adding extra I/O ports
- Software links