A yellow and white spiral

Description automatically generatedBlue text on a black background

Description automatically generated

A blue circle and number

Description automatically generated

A red letter m and a white background

Description automatically generated

|  |  |
| --- | --- |
| **OLIMEX Ltd.** 2 Pravda St., P.O. Box 237, Plovdiv 4000 BULGARIA | **Contact:** Mr. Tsvetan Usunov **Email:** [info@olimex.com](mailto:info@olimex.com) **Voice:** +359-32-626259, +359-32-267407, +359-32-621270 |

# Welcome – please read!

Welcome to the modern retro computer world, where you can experience the technology from the 70s and 80s, but with a modern spin on it!

This document covers both the Neo6502 and Neo6502pc computer. Detailed specifications and the differences between the two can be found in Appendix A.

|  |
| --- |
| Neither of the devices (the Neo6502 and Neo6502pc) are turn-key solutions. Both devices require intermediate electronics and computer use knowledge. While both devices have appeared in social media as an out-of-the-box video game platform, it will require that you read this document, so that you gain the best experience! |

## Please Note

A grey banner with white text

Description automatically generatedRegardless of the function you are hoping to utilize the Neo6502 or Neo6502pc, you must be familiar with the process of reprogramming (also known as flashing firmware) the 2MB flash memory utilized by the RP2040. The firmware defines what function the Neo6502 or Neo6502pc will perform. Current firmware available provide a BASIC interpreter (NeoBASIC) that is continues to be developed and improved, an Apple ][ emulator (using the real W6502), and an Oric Atoms. Other firmware is currently being developed, so explore the various user forums, Discord, and Facebook to discover the endless possibilities of the Neo6502 and Neo6502.

**Please read the Programming the RP2040 Section (page 5)**

Both devices require that you obtain or supply the following for proper operation:

**Neo6502**

* USB-C Power Source (5v, 0.2 amps).
* A USB cable with a USB-A on one end, and the appropriate end that will connect to your computer *(used to re-program the RP2040)*.
* *Optional,* enclosing case for the Neo6502, *available from Olimex.*
* *Optional*, USB-A Flash Drive (*highly recommend USB3, ~8 GB*).
* *Optional*, USB Hub (*Olimex USB-NeoHub is highly recommended for compatibility*).

**Neo6502pc**

* USB-C Power Source (5v, 1 amp).
* A USB cable with a USB-C on one end, and the appropriate end that will connect to your computer.
* USB-A Flash Drive (highly recommend USB3, ~8 GB).
* USB Keyboard *(wired and wireless w/USB dongle are supported).*
* *Optional,* USB Gamepad.

# Table of Contents

[Welcome – please read! 2](#_Toc174804094)

[Please Note 2](#_Toc174804095)

[Table of Contents 3](#_Toc174804096)

[About the Neo6502 5](#_Toc174804097)

[About the W65C02 processor 5](#_Toc174804098)

[Programming the RP2040 6](#_Toc174804099)

[Prerequisites 6](#_Toc174804100)

[RP2040 programming for the Neo6502 6](#_Toc174804101)

[RP2040 programming for the Neo6502pc 7](#_Toc174804102)

[Programming Troubleshooting 7](#_Toc174804103)

[Current Firmware 8](#_Toc174804104)

[NeoBasic (codename: Morpheus) 8](#_Toc174804105)

[Apple ][ and //e Emulation 8](#_Toc174804106)

[Apple ][ TotalReplay 9](#_Toc174804107)

[Oric Atmos 9](#_Toc174804108)

[NeoBASIC 10](#_Toc174804109)

[Programming Reference and Technical Documentation 10](#_Toc174804110)

[NeoBASIC Technical Reference 11](#_Toc174804111)

[Binary Operators 11](#_Toc174804112)

[Functions 12](#_Toc174804113)

[Arithmetic and Boolean Functions 12](#_Toc174804114)

[File System and I/O Functions 13](#_Toc174804115)

[BASIC Interpreter Functions 13](#_Toc174804116)

[String Functions 14](#_Toc174804117)

[Hardware Information Functions 15](#_Toc174804118)

[Commands 17](#_Toc174804119)

[Flow Control Commands 17](#_Toc174804120)

[File System and I/O Commands 19](#_Toc174804121)

[BASIC Commands 21](#_Toc174804122)

[Interfacing with hardware 24](#_Toc174804123)

[Graphics Commands 25](#_Toc174804124)

[Pixel Colors 26](#_Toc174804125)

[Sprite Commands 27](#_Toc174804126)

[MOS Commands 28](#_Toc174804127)

[The Inline Assembler 29](#_Toc174804128)

[[] Operator 29](#_Toc174804129)

[Zero-Page Usage 30](#_Toc174804130)

[Pascal for the Neo6502 32](#_Toc174804131)

[Appendix A 36](#_Toc174804132)

[Neo6502 37](#_Toc174804133)

[Hardware Pictures 37](#_Toc174804134)

[Neo6502pc 39](#_Toc174804135)

[Features 39](#_Toc174804136)

[Neo6502pc – Hardware Pictures 40](#_Toc174804137)

[Neo6502pc Specific Hardware Specifications 42](#_Toc174804138)

[Neo6502pc – Schematic 42](#_Toc174804139)

[Neo6502pc – 12 GPIO EXT1 Connector 42](#_Toc174804140)

[Shared Hardware 43](#_Toc174804141)

[Neo6502pc and Neo6502 – W6502 Bus Connector 43](#_Toc174804142)

[Neo6502pc and Neo6502 – UEXT Connectors 44](#_Toc174804143)

[Neo6502pc and Neo6502 – Configuration Switch Block 45](#_Toc174804144)

[Appendix B – CREDITS and LICENSE 46](#_Toc174804145)

[Appendix C – Document Revision History 46](#_Toc174804146)

[Appendix D – About Olimex 47](#_Toc174804147)

[Appendix E – Online Resources 48](#_Toc174804148)

# About the Neo6502

The Neo6502 is a standalone modern retro computer with a real W65C02 processor and RP2040 co-processor. This small device works 3-times-faster than any of the other recent 6502 competitors and 30-times-faster than 6502 based machines from the 1980s.

The “Neo” name was used two reasons: First it implies a modern design; Second came from the analogy with the movie The Matrix where the W65C02 lives in virtual world – thinking it has real memory, video and keyboard – however in reality it is all virtual and emulated with the RP2040.

Blue text on a black background

Description automatically generatedBoth the Neo6502 and Neo6502pc are open-source hardware (https://freedomdefined.org/OSHW), with all CAD files and firmware available to support the future development of software and enhancements to the hardware.

There are two models available:

* The Neo6502, an open circuit board computer (2 revisions, A & B).
* The Neo6502pc, a Neo6502 enclosed in a 3D-printed case with a LCD display, USB ports, UEXT and 6502 interface ports and more.

More technical specifications can be found in Appendix A (page 20). More information about the Neo6502 project, please refer to the Neo6502 website: <http://www.neo6502.com>

## About the W65C02 processor

The W65C02, being a more modern 6502 than the old retro metal oxide semiconductor chip – in that it can go much faster than was possible in the 1970s and 1980s. The W65C02 can even be overclocked to 16 MHz, but on the Neo6502 it is running at 6.25 Mhz, which is closer to the clock speed of the Amiga and Atari ST than the Atari or C64, and a lot faster than when most of the retro games were being coded.

The Neo6502 features a real W65C02S processor, which does all the computing with real timing versus emulation, but the real power of the machine coms from the RP2040 which provides the memory, video, keyboard input, and additional IO for SPI, I2C, UART, and so on.

Things like complex math (multiplication, floating-point) and graphics are also handled by the RP2040, acting like a co-processor. Unlike other similar architectures, the RP2040 has direct memory access (providing the memory for the 6502) so there are no additional big data transfers between the chips to wait for, making things all much more efficient.

The processor gets 64kb of RAM, but there is 2 MB of flash memory on board, access to USB flash drive for storage via USB or expansion port (for SD card support), and there is a 40-pin connector that offers up a bus of all the 6502 signals and pins that can be used to interface with or use for experiments. The UEXT ports already support quite a few modules from Olimex that support UEXT specification (<https://www.olimex.com/Products/Modules/>).

# Programming the RP2040

The process of programming the RP2040 is a fairly easy process, *however*, it has a very specific manner and steps that must be followed to have a successful reprogramming.

A close-up of a computer

Description automatically generated

**NOTES**

* Some emulators require all switches of the configuration switch block be in the on (closed) position.

## Prerequisites

* Your computer should be on, and you must be logged in and have the desktop present. Best experience comes with no CPU intensive tasks running on your computer.
* You have the latest version of the firmware that you want to use downloaded to your computer. *It is highly recommended that you download the firmware file from the “source of truth” (the developer’s Github repository or website).*A firmware file come in various sizes and names, based on the functionality it performs, however it will always have the uf2 file extension.
* Make sure the Neo6502 device has been powered down.

## RP2040 programming for the Neo6502

**Required hardware:**

* A computer with a USB port and a modern operating system.
* A Neo6502 computer.
* A USB cable with a USB-A on one end, and the appropriate end that will connect to your computer.

A close-up of a blue box

Description automatically generated**Steps:**

1. Connect the USB cable between your computer and the Neo6502 USB-A port. *If you have a USB hub connected or any other device connected to the USB-A port, please disconnect it during this process.*
2. Press and hold the "boot" button (bottom left, with the UEXT port on the left and the W6502 bus on the bottom). *Ensure you have heard or felt the button depress with a satisfying “click”.*
3. Turn the power on.
4. Release the "boot" button.
5. A volume will appear on your computer with the name “RPI-RP2”.
6. Copy the appropriate UF2 file to the “RPI-RP2” volume.
7. **Do not be alarmed**, as soon as the copy is finished, the volume will disappear. *This indicates that the firmware has been successfully uploaded and programming has begun and will only take a few seconds*.
8. Reconnect the USB hub and other devices that were removed on step 1.

## RP2040 programming for the Neo6502pc

**Required hardware:**

* A computer with a USB port and a modern operating system.
* A Neo6502pc computer.
* A USB cable with a USB-C on one end, and the appropriate end that will connect to your computer.

A close-up of a blue box

Description automatically generated**Steps:**

1. Connect the USB Cable between your computer and the Neo6502 USB-C port (with the LCD facing up, the USB-C port on the left).
2. Slide the programming switch on the back of the Neo6502pc to the programming position (with the switch facing up and in the upper left corner – move to the right-most position).
3. Press and hold the "boot" button (to the left of the programming switch). *Ensure you have heard or felt the button depress with a satisfying “click”.*
4. Continue to press the “boot” button and turn the power on.
5. Release the "boot" button.
6. A volume will appear on your computer with the name “RPI-RP2”.
7. Copy the appropriate UF2 file to the “RPI-RP2” volume.
8. **Do not be alarmed**, as soon as the copy is finished, the volume will disappear. *This indicates that the firmware has been successfully uploaded and programming has begun and will only take a few seconds*. The Neo6502pc will automatically reboot using the new firmware.
9. Move the programming switch back to “run” position.

**SUCCESS**

Based on the firmware that was just flashed, the Neo6502pc will now operate within the firmware function. Please refer to the documentation that comes with the firmware to know the next steps. The most popular firmware and their next steps are provided in this document.

### Programming Troubleshooting

* If you are using a Neo6502pc, ensure the programming switch in in the “program” position.

# Current Firmware

The following are accurate as of the August 4th, 2024 revision of this document.

## NeoBasic (codename: Morpheus)

Maintained by Paul Robson (paul@robsons.org.uk)

GitHub Repository: <https://github.com/paulscottrobson/neo6502-firmware>

Obtain the firmware from the repository link

1. A screen shot of a computer

   Description automatically generatedWithin the Github respository, navigate to the releases section (right side)
2. Click on the link (release number). This will take you to the releases list.
3. Locate and click the zip file to download it.
4. Unzip the file.
5. Locate the “firmware\_usb.uf2” file.   
   *The “*firmware\_sd.uf2*” file is used when you are using the SDCard adapter.*
6. Follow the directions above to program the RP2040 on page 4.

Please refer to the NeoBasic section (page 9) for more information.

## A blue screen with white text Description automatically generatedApple ][ and //e Emulation

Maintained by Veselin Sladkov (veselin.sladkov@gmail.com)

Obtain the firmware from: <https://github.com/vsladkov/reload-emulator>

The firmware source code is found on the repository; however, it is not compiled into a uf2 file. You can download the uf2 firmware file from Olimex’s FTP site: <https://ftp.olimex.com/Neo6502/>

1. Click the link to open the Olimex FTP site.
2. Click and download the “blank\_disk\_for\_apple2e\_code\_development\_apple2e\_ProDOS\_2\_4\_3.zip” file.
3. Unzip it, and copy the “ProDOS\_2\_4\_3.po” to a flash drive.
4. Follow the directions above to program the RP2040 on page 4, with the “apple2e.uf2” file.

You can replace the “ProDOS\_2\_4\_3.po” with other disk images that can be found on the internet. Check out the Apple ][ section on the Internet Archive (<https://archive.org/details/softwarelibrary_apple_games>) as well as other locations.

## Apple ][ TotalReplay

Maintained by Veselin Sladkov (veselin.sladkov@gmail.com)

Obtain the firmware from: <https://github.com/vsladkov/reload-emulator>

The firmware source code is found on the repository; however, it is not compiled into a uf2 file. You can download the uf2 firmware file from Olimex’s FTP site: <https://ftp.olimex.com/Neo6502/>



1. Click the link to open the Olimex FTP site.
2. Click and download two files:

* “Total Replay v5.1.hdv” file.
* “apple2e-5.uf2” file.

1. Copy the “Total Replay v5.1.hdv” to a flash drive.
2. Follow the directions above to program the RP2040 on page 4, with the “apple2e-5.uf2” firmware file.

If successful, turning on the Neo6502 device, will present you with the TotalReplay title screen. All games can be played with a keyboard and some games support the USB gamepad, your milage by vary.

## A screen with a white screen Description automatically generatedOric Atmos

Maintained by Veselin Sladkov (veselin.sladkov@gmail.com)

Obtain the firmware from: <https://github.com/vsladkov/reload-emulator>

The firmware source code is found on the repository; however, it is not compiled into a uf2 file.

You can download the uf2 firmware file from Olimex’s FTP site: <https://ftp.olimex.com/Neo6502/uf2/oric_960x540_372MHz.uf2>

This is an older version, and no compiled version with updated firmware is available as a download, as you must have copies of the Oric ROMs. You will need to compile it yourself or ask folks on social media if a updated compiled version is available.

A blue circle with black background

Description automatically generated

# NeoBASIC

*Written by Paul Robson*

NeoBASIC runs very fast compared to most 6502-based home computers back in the day that were 2 Mhz or less. Compared to the FPGA-assisted Commander X16 that runs at 8 Mhz, it underperforms versus the Neo6502 in this BASIC line draw demo, partially due to the inefficient BASIC interpreter.

You can program right on the Neo6502 using NeoBASIC, or you can use the web-based Neo6502 Basic emulator available at neo6502.com. There are also emulators that will run on Windows, Linux, and MacOS desktop operating systems. Cross-development is possible even with NeoBASIC by using the Python script to tokenize your basic listing ready for execution.

There are some NeoBASIC based games that have been written that really show off some of the very powerful features:

|  |  |
| --- | --- |
| * **Galaxians** arcade game written by Paul Robson *Still a work-in-progress* * **Frogger** arcade game * written by Paul Robson *Still a work-in-progress* | * **Space Invaders** arcade game |

## Programming Reference and Technical Documentation

The information that follows describes the NeoBASIC interpreter. It is for experienced programmers and includes explanations of all BASIC statements and functions, memory tables, sprite usage, and file formats.

A black and white logo

Description automatically generatedPlease note that many BASIC commands are just wrappers of specific API functions  
(noted with the API logo)

If you have never used BASIC, read the introductory topics before you begin. Before using any statement or function, thoroughly read the description.

# NeoBASIC Technical Reference

## Binary Operators

|  |  |  |
| --- | --- | --- |
| Precedence | Operator | Notes |
| 4 | \* | Multiplication operator |
| 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 | + | Addition operator |
| 3 | - | Subtraction operator |
| 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 |

## Functions

### Arithmetic and Boolean Functions

|  |  |
| --- | --- |
| Function | Description (*function and return value)* |
| atan(n) | Calculate the arctangent (the inverse tangent function) of n in degrees | |
| atan2(y,x) | Calculates the arctangent (the inverse tangent function) of y,x  if x equals 0, atan2 returns π/2 if y is positive, -π/2 if y is negative, or 0 if y is 0. | |
| cos(n) | Calculate the cosine of n (n must be in degrees) | |
| exp(n) | Calculates the exponential value of a floating-point argument n (en, where e equals 2.17128128...) | |
| FALSE | Return constant 0, improves boolean readability | |
| int(n) | Return the whole part of the float value n. Integers are unchanged. | |
| log(n) | Calculate the natural logarithm (e.g. ln2) of n. | |
| max(a,b) | Return the largest of a and b (numbers or strings) | |
| min(a,b) | Return the smallest of a and b (numbers or strings) | |
| pow(a,b) | Returns a raised to the power b; the result is always floating point. | |
| rand(n) | Returns a random integer, where 0 < x < n. *The value returned will be between 0 and n-1.* | |
| rnd(0) | Returns a random number where 0 < x < 1. The value (zero) passed is ignored. | |
| sin(n) | Calculate the sine of n (n must be in degrees) | |
| sqr(n) | Calculates the square root of n | |
| tan(n) | Calculates the tangent of n (n must be in degrees) | |
| TRUE | Return constant -1, improves boolean readability | |

### File System and I/O Functions

|  |  |
| --- | --- |
| Function | Description (*function and return value)* |
| eof(f) | Returns non-zero value if at end of file f. |
| exists(file$) | Returns true (-1) if the file exists, false (0) otherwise |
| locale a$ | Sets the locale to the ISO 3166-1 alpha-2 country code a$ e.g. locale "de" for Germany, “us” for United States of America. |
| mos(command)[[1]](#footnote-1)† | Returns zero if the command was successful, and non-zero error if it was unsuccessful or generated an error.  **See**: MOS Commands (page 28) |

### BASIC Interpreter Functions

|  |  |
| --- | --- |
| Function | Description (*function and return value)* |
| err | Current error number |
| erl | Current error line number |

### String Functions

|  |  |
| --- | --- |
| Function | Description (*function and return value)* |
| asc(s$) | Return ASCII value of first character or zero for empty string. |
| chr$(n) | Convert ASCII decimal value to a character. |
| instr(str$,search$) | Returns the first position of search$ in str$, indexed from 1. Returns zero if not found. |
| isval(s$) | Converts string to number, returns -1 if okay, 0 if fails. |
| left$(a$,n) | Left most number of characters (n) of a$. |
| len(a$) | Return length of string in characters. |
| lower$(a$) | Convert a string to lower case. |
| 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. |
| right$(a$,n) | Rightmost n characters of a$. |
| spc(n) | Returns a string containing the number (n) spaces. |
| str$(n) | Convert a number (n) to a string. |
| upper$(a$) | Convert a string to upper case. |
| val(s$) | Convert string to number, error if bad number. |

### Hardware Information Functions

|  |  |
| --- | --- |
| Function | Description (*function and return value)* |
| alloc(n) | Allocate n bytes of memory, return address |
| analog(n) | Read voltage level on pin n -- returns a value from 0 to 4095 |
| deek(a) | Read word value at a |
| event(v,r) | event takes an integer variable and a fire rate (r) in 1/100 seconds, 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 resynchronize. |
| 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 |
| 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. |

**Hardware Information Functions** (*continued)*

|  |  |
| --- | --- |
| Function | Description (*function and return value)* |
| key(n) | Return the state of the given key. The key is the USB HID key scan code n. |
| mouse(x,y[,scroll]) | Reads the mouse. The return value indicates button state (bit 0 left, bit 1 right), and the mouse position and the scrolling wheel position are updated into the given variables. |
| 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. |
| spoint(x,y) | Reads the color index on the sprite layer. 0 is transparency |
| tab(n) | Advance to screen column n if not past it already. |
| time() | Return time since power on in 1/100 seconds. |
| uhasdata() | Return true if there is data in the UART Receive buffer. |
| vblanks() | Return the number of vertical-blanks since power on. This is updated at the start of the vertical-blank period. |

## Commands

### Flow Control Commands

|  |  |
| --- | --- |
| Control Structure | Description |
| do  ...  *exit*  ...  loop | Provides an infinite loop structure, with the ability to use the exit command to break out of the loop at any point.  10 index = 0  20 do  30 print “Hello World!”  40 index = index + 1  50 if index > 10 then exit  60 loop |
| end | End the current running program |
| for {var} = {start} to/downto {end}  ...  next {var} | Provides a controlled loop (for / next) using a range of values. *Note this is non-standard for/loop control, as there are limitations.* Limitations:   * The index must be an integer. * The step is controlled used “to” (1) or “downto” (-1).   next cannot specify an index and cannot be used to terminate loops (*using the wrong index).* Execution must operate in order and flow cannot be stopped arbitrarily. The variable after next is ignored.  10 for count = 10 downto 1  20 print count  30 next count  40 print “Blast off!” |
| gload {filename} | Load filename into graphics memory. |
| gosub {expr}[[2]](#footnote-2)† | Call subroutine at line number. A return command will return control to the line after the gosub call. |
| goto {expr}[[3]](#footnote-3)† | Transfer execution to a line number. |
| return[[4]](#footnote-4)† | *See gosub.* |

**Flow Control Commands** (*continued…)*

|  |  |
| --- | --- |
| Control Structure | Description |
| if {expr} then *nn* | Provide a single conditional, where execution is transferred to line number nn if the {expr} is true.  **NOTE:** if {expr} goto nn does not work. |
| if {expr}:  ..  else  ..  endif | Provides an if-then-else condition. More than one line can be used between if and endif or between if and else and else and endif. *The else clause is optional.*  Code between if and else will only execute if the {expr} evaluates as true, and the code between else and endif will only execute if the {expr} evaluates as false. |
| on error *nn* | Install an error handler, when execution is transferred to line number nn when an error occurs (effectively a goto command). |
| repeat  ..  until {expr} | Provides a finite loop structure that will end when the expression following the until evaluates as true, else it will repeat while the expressing following the until evaluates as false. |
| run | Will execute the program in memory at the first and lowest line number. |
| stop | Will terminate the program with an error. |
| wait {s} | Waits for {s}, where {s}is 1/100 seconds. |
| while {expr}  ...  wend | Provides a finite loop structure that will end when the expression following the while evaluates as false, else it will repeat while the expressing following the while evaluates as true. |

### File System and I/O Commands

|  |  |
| --- | --- |
| Command | Description |
| cat "*{pattern}"* | Show contents of current directory of the attached storage device.  Can take an optional string {pattern} which only displays filenames containing those characters, example cat "ac" will only displays files with that contains the substring ac. |
| close *[handle]* | Close a file by handle.  The handle is optional, and if not provided, all files will be closed. |
| input {var} | Will wait for input that ends when hitting return.  The content enter will be stored in the variable {var}. |
| input #{channel},{var},{var} | Reads a sequence of variables from the open file. |
| ireceive {d},{a},{s} | Receive bytes starting at a, count s to or from device d. |
| itransmit {d},{a},{s} | Send bytes starting at a, count s to or from device d. |
| isend {device},{data} | Send data to i2c {device}; this is comma separated data, numbers or strings. If a semicolon is used as a separator e.g. 4137; then the constant is sent as a 16-bit value. |
| iwrite {dev},{reg},{b} | Write byte to I2C Device Register |
| load "file"[,{address}] | Load file to BASIC space or given address. |
| mos "command" | Execute MOS command within the quotes.  **See**: MOS Commands (page 28) |

**File System and I/O Commands** (*continued…)*

|  |  |
| --- | --- |
| Command | Description |
| 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. |
| print {string | var} | This will output the contents that following the command. A string is encapsulated in double-quotes. A var can be a string or number variable.  You can concatenate strings and vars together with the + symbol. If you end a print statement with a semi-colon, then the next print will follow at the end of this print, effectively appending. |
| 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 separated by character code 10). This can be mixed with the above format *but* the sequence has to be the same ; you can't write a string using print line and read it back with input and vice versa. All variables must be strings. |
| run "{program}" | Load & Run program. *The last quotation mark is optional.* |
| save "file"[,{adr},{sz}] | Save BASIC program or memory from {adr} length {sz} |
| sreceive {a},{s} | Receive bytes starting at a, count s to SPI device |
| stransmit {a},{s} | Send bytes starting at a, count s to SPI device |
| ssend {data} | Send data to SPI device; this is comma separated data, numbers or strings. If a semicolon is used as a separator e.g. 4137; then the constant is sent as a 16-bit value. |
| ureceive {d},{a},{s} | Receive bytes to/from the UART starting at a, count s |
| utransmit {d},{a},{s} | Send bytes to/from the UART starting at a, count s |
| usend {device},{data} | Send data to UART ; this is comma separated data, numbers or strings. If a semicolon is used |

### BASIC Commands

|  |  |
| --- | --- |
| Command | Description |
| ' {string} | Comment. This is a string for syntactic consistency. The tokenizer 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. |
| cls | Clear the graphics screen to current background colour. This does not clear sprites. |
| cursor {x},{y} | Set the text cursor position to position x,y on the screen. |
| 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 |
| edit | Basic Screen Editor |
| fkey | Lists the defined function keys |
| fkey {key},{string} | Define the behavior of F1..F10 -- the characters in the string |
| ink fgr[,bgr] | Set the ink foreground and optionally background for the console. |
| let {var} = {expr} | Assignment statement. The LET is optional. |

**BASIC Commands** *(continued…)*

|  |  |
| --- | --- |
| Command | Description |
| library | Librarise / Unlibrarise code. |
| list [{from}][,][{to}] | List program to display by line number or procedure name. |
| list {procedure}() |  |
| local {var},{var} | Local variables, use after PROC, restored at ENDPROC variables can |
| 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 |
| old | Undoes a new. This can fail depending on what has been done since the 'new'. |
| 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 |
| 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. |

**BASIC Commands** *(continued…)*

|  |  |
| --- | --- |
| Command | Description |
| 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. |
| restore | Restore data pointer to program start |
| restore {line} | Restore data pointer to line number |
| 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 |

### Interfacing with hardware

|  |  |
| --- | --- |
| Command | Description |
| 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. |
| doke {addr},{data} | Write word to address |
| mon | Enter the machine code monitor |
| pin {pin},{value} | Set UEXT {pin} to given value. |
| pin {pin} INPUT | output |
| poke {addr},{data} | Write byte to address |
| sys {address} | Call 65C02 machine code at given address. Passes contents of variables A,X,Y in those registers. |
| uconfig {baud}[,{prt}] | Set the baud rate and protocol for the UART. Currently only 8N1 is supported. |

### Graphics Commands

|  |  |
| --- | --- |
| Command | Description |
| 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 | Modifier to a graphic command to change what color c the command will use. |
| ink a,x | Draw by anding the color with a, and xoring it with x. |
| solid | Fill in rectangles and ellipses. For images and text, forces black background. |
| text {str} to x,y | Draw/place text at t c0dc0he specified x,y coordinates. text “Hello World!” to 20,20 |
| rect | rect {*solid* | *frame*} x1,y1 to x2,y2  Will draw a rectangle with the upper-left-corner (x1,y1) to the bottom-right-corner (x2,y2).  **Note:** Once used, rect and ellipse commands will continue using the solid or frame state until changed.  **See** frame, solid |
| ellipse | ellipse {*solid* | *frame*} x1,y1 to x2,y2 |
| frame | A modifier for rectangles and ellipses, where it will only draw the outline, not filling it in.  rect frame x1,y1 to x2,y2 will draw an empty rectangle. |
| solid | A modifier for rectangles and ellipses, where it will only draw a filled in rectangle or ellipse.  rect solid x1,y1 to x2,y2 will draw a filled in rectangle. |

|  |  |
| --- | --- |
| dim n | Set the scaling to **n** (for text, image, and tilemap only), and must be an integer.  **text "Hello" dim 2 to 10,10 to 10,100** will draw the word “Hello” at double its size!  Tiles can only be scaled at 1 or 2 (when scaling at 2, tiles are drawn at a size of 32x32, versus the scale 1 of 16 x16). |
| move |  |
| plot |  |
| line |  |
| flip |  |
| anchor |  |
| image |  |
| sprite |  |

### Pixel Colors

*Color swatches are where taken from the built-in LCD display of the Neo6502pc.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pixel | Color |  | Pixel | Color |  |
| 0 | Black *Transparent* | A yellow rectangular device with a blue screen  Description automatically generated | 8 | Black | A yellow rectangular device with a screen  Description automatically generated |
| 1 | Red | A screen with a red circle on it  Description automatically generated | 9 | Dark Grey | A yellow rectangular device with a white circle on it  Description automatically generated |
| 2 | Green | A yellow rectangular screen with a green circle on it  Description automatically generated | 10 | Dark Green | A yellow rectangular device with a green circle on it  Description automatically generated |
| 3 | Yellow | A yellow rectangular screen with a yellow circle and text on it  Description automatically generated | 11 | Orange | A yellow rectangular screen with a yellow circle and white text  Description automatically generated |
| 4 | Blue | A yellow rectangular device with a blue circle on it  Description automatically generated | 12 | Dark Orange | A yellow screen with a message on it  Description automatically generated |
| 5 | Magenta | A yellow rectangular device with a purple circle on it  Description automatically generated | 13 | Brown | A screen with text on it  Description automatically generated |
| 6 | Cyan | A yellow rectangular device with a blue circle on it  Description automatically generated | 14 | Pink | A yellow rectangular device with a white circle on it  Description automatically generated |
| 7 | White | A yellow rectangular device with a white circle on it  Description automatically generated | 15 | Light Grey | A yellow rectangular screen with a blue circle on it  Description automatically generated |

### 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)

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.

### MOS Commands

Using the MOS commands to access OS disk/file functionality.

MOS commands can be used in for different ways:

* On the NeoBASIC command line, prefixing the command with an asterisk.  
  **Example**: \*del myfile.txt
* On the NeoBASIC command line, use the mos BASIC command. Surround the MOS command in quotes.  
  **Example**: mos "del myfile.txt"
* Within a NeoBASIC program, you can use the mos BASIC function. Surround the MOS command in quotes.  
  **Example**: if mos("del myfile.txt") > 0 then …
* Using the API function, which take two strings as parameters.  
  **Example**: ????

**Available MOS Commands**

|  |  |  |  |
| --- | --- | --- | --- |
| cat | Prints out the file listing of the current directory. | ren | Will rename a file or directory. Works identically as the copy command. |
| del | Delete the specified file. Can be used to delete a directory, only if it is empty. *If a directory is not empty, the command generates an error.* | md | Make a directory. Specify the name of the directory you want to create. |
| copy | Copy on path to another.  Requires two parameters, the first is the source path and file, and the second is the destination path and file.  If no path is given, but different names are given, then you get a copy of the source file specified in the first parameter in the current directory.  You cannot make copies of directories. | cd | Change Directory.  Specify the name of the directory that you wish to traverse. |
| file | Will verify that a file (*not a directory*) exists. Return zero if it exists, and non-zero if it does not exist. |

**Standard Unix POSIX paths are used with the MOS commands.**

|  |  |
| --- | --- |
| ./ | Current directory |
| ../ | Go up in the hierarchy, relative to the current directory (*sometime referred as the going back a directory*). |
| / | The root or top-level directory. |

## 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 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. For example, unlike on the Commodore 64, 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.

# Pascal for the Neo6502

# Appendix A

This appendix offers the detailed specifications for each of the Neo6502 computers and the differences between them.

|  |  |  |
| --- | --- | --- |
| Feature | Neo6502  Computer | Neo6502pc Computer |
| Physical W65C02 processor running at 6.25Mhz | Yes | Yes |
| RP2040 SoC (System on a Chip) w/ 2MB Flash | Yes | Yes |
| 10-pin UEXT connector | 1x | 4x |
| 6502 bus connector | Yes | Yes |
| Audio mini speaker | Yes | Yes |
| Audio 3.5mm connector | Yes | Yes |
| USB-C power supply connector | Yes | Yes |
| DVI/HDMI connector | Yes | No[[5]](#footnote-5) |
| USB-A Port[[6]](#footnote-6) | 1x[[7]](#footnote-7) | 3x |
| 4-position configuration slide switch | Yes | Yes |
| Boot Button | Yes | Yes |
| One 14-pin external 12 GPIO connector | No | Yes |
| Programming slide switch | No | Yes |
| USB-C Programming Port | No | Yes |
| Build-in LCD display w/ touch panel acting like mouse | No | Yes |
| Build in LiPo battery w/ charger circuit and battery monitoring | No | Yes |
| Power on off switch | No | Yes |

## A red circuit board with black and red components Description automatically generatedNeo6502

The Olimex Neo6502 was the first model released. The main components are a W65C02 and a Raspberry Pi RP2040. The W65C02 runs the machine code (at about 6.3Mhz), with the RP2040 providing RAM, Video and other aspects.

A close-up of a circuit board

Description automatically generatedEarly adopters had a revision A board (purple), and later the revision B board (red) was released. Both are almost identical, with the exception that the revision A board required a couple of wires connecting pins of the UEXT to the 6502 bus to work properly. Both models come as the board only, with cases available to protect the board.

### Hardware Pictures

A red circuit board with white text

Description automatically generated

## Neo6502pc

The Olimex Neo6502pc is an open-source hardware and software standalone modern retro computer with a real W65C02 processor, RP2040 co-processor, USB host (with 3x USB-A ports), Lithium polymer (Lipo) battery with charging circuit and battery monitoring.

The design goal with Neo6502pc was to create a simple retro-computer with a real 6502 processor that has modern features (provided by the Raspberry RP2040 co-processor) such as HDMI video output, USB ports that support USB keyboards, flash drives for storage, and USB game pad. The RP2040 co-processor emulates the RAM for the W6502 processor.

More Information: <https://www.olimex.com/Products/Retro-Computers/Neo6502pc/open-source-hardware>

## Features

* A real W65C02 processor clocked at 6.25Mhz
* Graphics co-processor RP2040 providing 320 x 240 resolution with 256-color display on HDMI/DVI.
* 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
* Four UEXT interface ports to access a wide range of hardware add ons.
* 1 channel "beeper" sound with SFX library (to be replaced by AY-3-8910 Emulation)
* USB flash drive support for storage w/ optional SD Card support.
* Supports standard USB keyboard.
* 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
* Huge open-source community that has written documentation, provided samples, and code including games.
* 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 - 64KB linear RAM space for code
* LCD display
* Internal battery backup power supply which allows it to operate up to 3 hours without external power supply
* Three external and one internal USB hosts (internal is connected to LCD touch panel)
* Audio output
* 12 GPIO extension connector
* USB-C for power and internal battery charging.
* Second USB-C for RP2040 firmware programming
* Dimensions 220 x 130 x 35 mm

### Neo6502pc – Hardware Pictures

A blue rectangular device with a screen and buttons

Description automatically generatedA blue rectangular object with red text

Description automatically generated

A blue rectangular object with red text

Description automatically generated

**Neo6502pc – Hardware Pictures** *(continued)*

|  |  |  |
| --- | --- | --- |
| A red rectangular device with a screen  Description automatically generated | A black rectangular device with a screen  Description automatically generated | A yellow rectangular device with a screen  Description automatically generated |
|  |  |
| A green rectangular device with a screen  Description automatically generated | A blue rectangular device with a screen  Description automatically generated |

A yellow box with wires and a blue object

Description automatically generated with medium confidence

## Neo6502pc Specific Hardware Specifications

### Neo6502pc – Schematic

The latest schematic for the Neo6502pc is available on GitHub using this link: <https://github.com/OLIMEX/Neo6502pc>

### Neo6502pc – 12 GPIO EXT1 Connector

Neo6502pc has a CH32V003 expander IC which is connected to RP2040 via I2C and can monitor battery charge, the presence of the external power supply and access to the 12 GPIOs via the EXT1 connector:

|  |
| --- |
| A diagram of an electrical wiring  Description automatically generated |
| Neo6502pc and Neo6502 – 12-pin GPIO EXT1 connector schematic |

# Shared Hardware

## Neo6502pc and Neo6502 – W6502 Bus Connector

All 6502 signals are available on BUS1 connector for attaching external hardware on it. Signals available:

* +5V
* 3.3V
* GND
* D0-D7
* A0-A15
* PHI2
* R/W
* RESB
* SOB
* MLB
* VPB
* SYNC
* NMIB
* IRQB

Two signals of RP2040 SWDIO and SWCLK are also present for RP2040 debugging, these should not be connected on the external 6502 peripheral boards.

|  |
| --- |
| A diagram of a bus  Description automatically generated |
| Neo6502pc and Neo6502 – W6502 bus connector schematic |

## Neo6502pc and Neo6502 – UEXT Connectors

UEXT (Universal EXTension) connectors have the following signals available. All signals are with 3.3V levels.

* +3.3V
* GND
* I2C
* SPI
* UART

UEXT connector can be found in many different shapes, however the connector used on the Neo6502pv uses a UEXT connector that is 0.1” 2.54mm step boxed plastic connector.

|  |
| --- |
|  |
| Neo6502 and Neo6502pc UEXT Connector w/signal identification |

Olimex has developed number of [MODULES](https://www.olimex.com/Products/Modules/) with this connector: temperature, humidity, pressure, magnetic field, light sensors, LCDs, LED matrix, Relays, Bluetooth, Zigbee, WiFi, GSM, GPS, RFID, RTC, EKG, sensors and etc.

Neo6502pc UEXT connector is wired to RP2040 GPIOs as follows:

|  |
| --- |
| A diagram of a circuit  Description automatically generated |
| Neo6502 and Neo6502pc UEXT schematic |

## Neo6502pc and Neo6502 – Configuration Switch Block

The Slide configuration switch can enable/disable the Buzzer, also can connect or disconnect RESB, NMIB and IRQB to RP2040 UEXT signals.

|  |
| --- |
| A diagram of a circuit board  Description automatically generated |
| Neo6502 and Neo6502pc configuration switch block schematic |

A close-up of a computer

Description automatically generated

By default, the Neo6502pc is shipped with all switches in the closed state – enabling the Piezoelectric Buzzer Speaker, and all signals wired to the RP2040. With SW2, SW3 and SW4 enabled, the SPI on UEXT cannot be used.

# Appendix B – CREDITS and LICENSE

The Neo6502 board, schematics, and firmware are all part of an Open Source project created by Mr. Tsvetan Usunov of Bulgaria.

This document is an open-source copy-left document that is a collection and combination of various other open-source documents and is meant to be a superset of these documents and replaces them. Permission is hereby granted, free of charge, to any person obtaining a copy of these documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

\* All derivatives of this document must also carry the same open-source copy-left license.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

**This license can also be found in the Github repository:**  
https://github.com/jewettg/Neo6502-Documentation/blob/main/LICENSE.md

**This document initially created by**

**Greg Jewett**  
**Email:** greg(at)ejewett(dot)com  
**Website:** <https://sites.google.com/ejewett.com/gregjewett/home>

For a list of contributing authors and updates, please visit the Github repository:

https://github.com/jewettg/Neo6502-Documentation/blob/main/CONTRIBUTING.md

# Appendix C – Document Revision History

Please visit the GitHub repository file CHANGELOG.md

<https://github.com/jewettg/Neo6502-Documentation/blob/main/CHANGELOG.md>

# Appendix D – About Olimex

A red letter m and a white background

Description automatically generated

|  |  |
| --- | --- |
| **OLIMEX Ltd.** 2 Pravda St., P.O. Box 237, Plovdiv 4000 BULGARIA | **Contact:** Mr. Tsvetan Usunov **Email:** [info@olimex.com](mailto:info@olimex.com) **Voice:** +359-32-626259, +359-32-267407, +359-32-621270 |

Olimex Ltd is a leading provider for development tools and programmers for embedded market.

The company has 28 years of experience in designing, prototyping and manufacturing printed circuit boards, sub-assemblies, and complete electronic products.

We were established in 1991 in Plovdiv - the second largest city in Bulgaria.

We have extensive knowledge in analog, digital, and microcontroller design, and we offer our own-designed development boards, programmers and emulators for rapid prototyping ARM, AVR, MSP430, MAXQ and PIC microcontrollers.

Olimex is recognized as an approved third-party hardware developer by Texas Instruments Inc., Maxim Integrated, Atmel Inc., NXP Inc., ST Microelectronics Inc., IAR Systems AB, Cirrus Logic Inc., OKI Semiconductor Inc, Energy Micro Inc., and Microchip Inc.

We have over 30,000 active customer accounts who regularly use our services for electronic boards development and prototyping. Our design capabilities are backed by our own PCB prototype production and assembly facility, so all designs made by us are created with design-for-manufacturing in mind - which guarantees that they are optimized for reliability and provide cost-effective solutions for our customers.

The company’s 5,000 sq m. production buildings are situated on our 10,000 sq m. property.

# Appendix E – Online Resources

As an open-source project, there are a ton of resources already available for the Neo6502! Below is a list (not exhaustive, and growing) of various websites, repositories, and more.   
  
If you find a link that no longer works, please let us know.

|  |  |
| --- | --- |
| Blue text on a black background  Description automatically generated | * [Tindie (Australia)](https://www.tindie.com/products/agon/neo6502-rev-b-available-now/) * [The Pi Hut (UK)](https://thepihut.com/products/olimex-neo6502) * [Digikey (US)](https://www.digikey.com/en/products/detail/olimex-ltd/NEO6502/22078296) |
| GitHub logo PNG transparent image download, size: 1125x417px | **Neo6502:** [Olimex Neo6502 Github Repository](https://github.com/OLIMEX/Neo6502/)  **Documentation:** [Neo6502-Documentation](https://github.com/jewettg/Neo6502-Documentation) |
| A red letter m and a white background  Description automatically generated | **Neo6502 Project:** [Official Product Website](https://www.olimex.com/Products/Retro-Computers/Neo6502/open-source-hardware)  **UEXT Modules:** [UEXT Modules available from Olimex](https://www.olimex.com/Products/Modules/) |
| A blue text on a black background  Description automatically generated | [Facebook Page](https://www.facebook.com/groups/745798620676673/) |
| A blue text on a black background  Description automatically generated | [Discord Server](https://discord.com/invite/Z74ZJ7VMjD) |

1. † “MOS” or “mos” stands for Machine Operating System, which is BASIC command to access API functionality for OS commands. [↑](#footnote-ref-1)
2. † Provided to use only for porting code to the NeoBASIC. Continued use is not recommended as line numbers can change. The renumber command will change line numbers and not update gosub and goto calls. [↑](#footnote-ref-2)
3. [↑](#footnote-ref-3)
4. [↑](#footnote-ref-4)
5. The HDMI port is utilized by the built-in LCD display. [↑](#footnote-ref-5)
6. The USB-A port can be used for various USB accessories (keyboards, flash drives, gamepad, etc..) and for the Neo6502 – it serves as the RP2040 programming port (*requires a USB-A to USB-A cable*). [↑](#footnote-ref-6)
7. Requires the USB-Neohub (https://www.olimex.com/Products/USB-Modules/USB-NeoHub/open-source-hardware) the expand the number of USB-A ports available. Additional ports are required to utilize NeoBasic, as you need at minimum a flash drive and keyboard. *Unlike the USB-Neohub, not all USB hubs are compatible or supported*. [↑](#footnote-ref-7)