Neo6502 Outline Specification

17th November 2023. Outline Revision 1

All suggestions are welcome to paul@robsons.org.uk or Discord

This is a discussion document not a final statement.

Speed

Currently thanks to Rien Matthijsse and Sascha Schnieders work and assistance in figuring out why BRA and *only* BRA didn't work (!) this is running stable at about 3.5Mhz. It may be possible to speed it up.

Clean Machine

The plan is to have a largely empty machine in terms of software, rather like the Japanese Sharp series of Microcomputers, which came with a 4k ROM providing basic functionality and a lot of memory.

This was probably more useful than a ROM based machine ; the downside was that you had to load BASIC from tape. However, that is no longer an issue :-)

So one should be able to load EHBasic, MSBasic or any other language or tool that can be ported and have the maximum amount of memory, maybe 60k or 62k.

Boot ROM

The boot ROM would contain some common functionality, such as keyboard, character I/O and hopefully basic flash storage. As people seem to like it, I would add Wozmon (rewritten from the spec as I'm not sure about ownership).

This would still actually be RAM so if someone wants to implement CPM/65 or whatever they can simply overwrite it and talk to the hardware directly. It would be more accurate to call it 'loaded at Reset'.

Hardware communication

I plan to keep Rien Matthijsse's design of communication via memory. It does appear to work fine with simply calling code in the write update routine, but I will stick with the use of a control byte. I will move it from its current location ($D0xx) the primary reason that it's in the middle of what is now RAM.

8 bytes of the ROM space - which isn't actually ROM will be used for hardware interfacing. Byte 0 will be the communication byte, commands set this to non-zero, and the system sets it back to zero to acknowledge completeness. This will be placed somewhere close to the end of memory.

Hardware Devices

Screen

I plan to use the 320x240x256 mode which currently exists in Pico_v4 and is part of the Adafruit GFX Library. “Text mode” will use 5x7 ASCII characters with solid foreground and background, allowing for a 53x30 character display (this is built into the Adafruit library).

The library also provides the usual set of drawing functions, including Bitmaps and Sprites, and these can probably be added to at will.

It also allows the possibility of using the palette to effectively create two 4 bit planes, which may optimise sprites. (e.g. 0000cccc is the lower bit plane ccccxxxx is the upper bit plane if cccc is non-zero). The RP2040 may be fast enough to draw sprites on its own of course, I haven’t looked at the code or tried it out.

There won’t be direct access to video memory. Probably.

Note: needs to allow for different screen resolutions subject to memory and processor time.

Keyboard

There are two independent keyboard systems. One provides a keyboard press queue for typing stuff. The other provides the ability to read the current state of the keys so they can be used for game controllers. The keyboard isn’t actually working yet – I haven’t tried yet, all my time has been spent getting the 65C02 stable.

Internationalisation will be handled by the BIOS ROM.

Mouse

Well, I’m guessing here this would work ? If so, provide a pointer and interface. Would require a hub.

Sound

Sound is quite limited by the hardware (down to the lack of pins !), and the code for this already exists. The interface will allow for multiple channels just in cases.

File I/O

File I/O will be done almost entirely off 6502 ; the functions will just indicate to the RP2040 to ‘read/write this memory here’, stubs will be provided for serial I/O. Initially this will read and write to the unused flash. (A high priority will be using a USB key or similar for storage which would require a hub).

A consequence of this *I think* is that it will behave like a ZX81. The screen is maintained by one of the processor, and its code is read through a cache, and I think that is disabled by the flash writing routine. Which will mean a loss of sync.

Maths Coprocessor

One of the biggest slowdowns for an interpreted language (or a compiled one for that matter) is multibyte arithmetic. Especially floating point. It makes the CBM Disk Drives look fast.

To get round this I propose a stack based integer/floating point Maths Co-Processor which is available to everyone. This will make the new BASIC faster (and significantly smaller !) and those who think it's a bit of a cheat (and they have a point) can simply ignore it.

Turtle Graphics

Just in case someone would like this for education, for which it's way more suited than certain other hardware that shall remain nameless, or for that matter the ruddy Microbit (which isn't that much cheaper) I'd like to add a simple turtle graphics system by default.

Others

Provision for system timers. These will probably be directly accessible rather than accessed via the hardware interface.

Access to the UEXT bus so we can access Olimex's hardware is a fairly high priority option.

Emulator

I will write an emulator which will hopefully be in sync with the hardware most of the time. (cross platform).

Note

All code where possible will be open source MIT licensed so you can do what you want with it.

Where possible means everything I write and I would encourage anyone who writes for the system to do the same (or some other non-restrictive license, I like MIT because you can pretty much do anything you like with it)

Revisions

Revised	Notes
17/11/2023	Initial document by Paul Robson
	Communication area moved from page 0, tweaks.