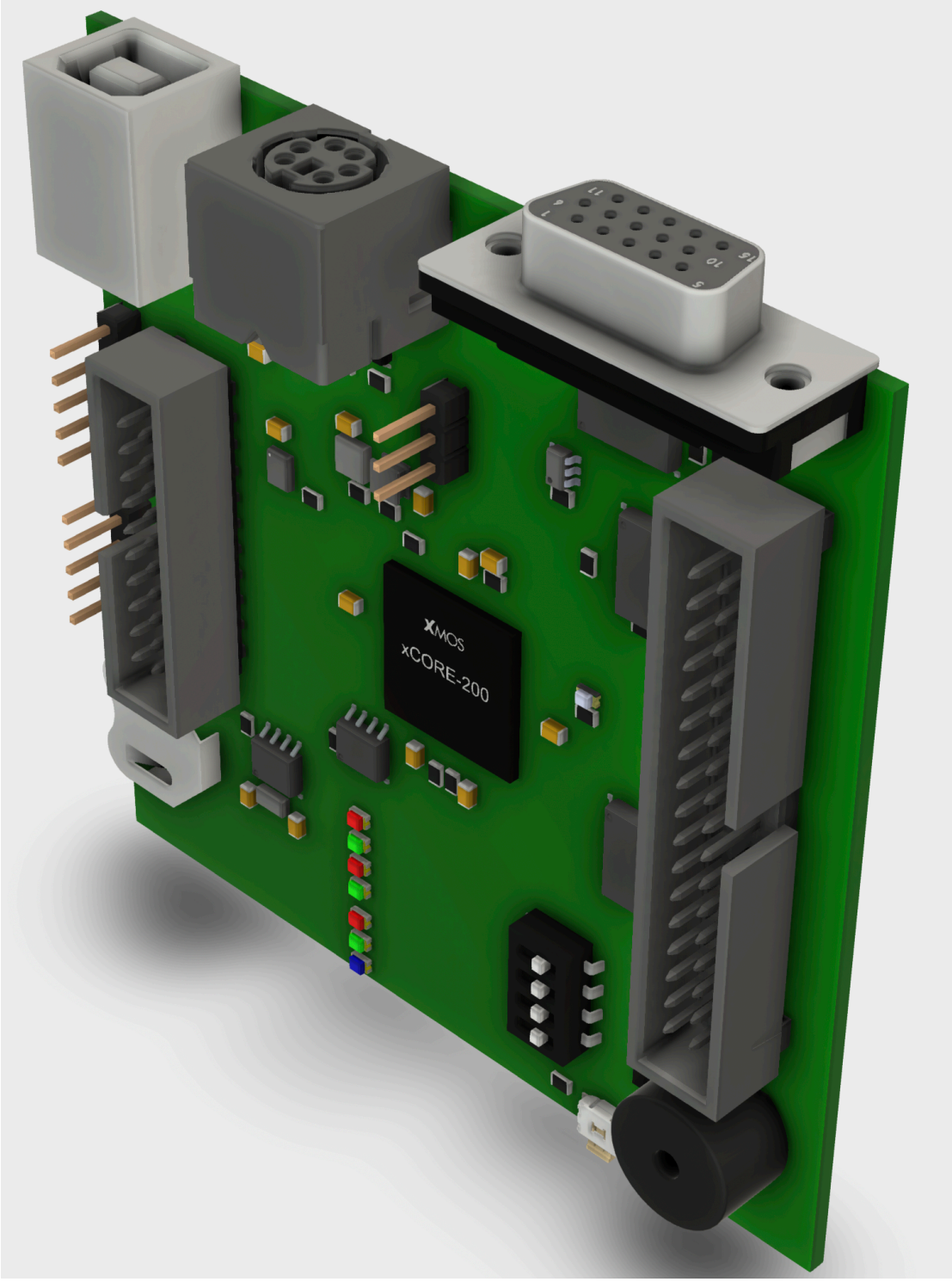


Microbox 2K2 User Guide

```
+++ Mon09 Ver 6.0  D.A.Rumball 2020 +++
Booting internal FLEX....
6809 FLEX V3.01

15:20:19  Wednesday 4th March 2020
SYSTEM DRIVE IS #0
WORK DRIVE IS #1
Microbox ][ auto setup of ASN and TTYSET.
+++
+++allocate
Drive 0 is the PROMdisk
Drive 1 is the RAMdisk
Drive 2 is FlexNet
Drive 3 is unassigned
+++
```

CONTENTS

Acknowledgements

I'd like to acknowledge and say thank you to all those in the FLEX User's Group who have been preserving and documenting these early 6809 FLEX based systems and especially Michael Evenson for the NetPC/FLEXNet protocols and utilities used in the development of the MB2K2 and Hermann Seib for the A09 assembler.

D.A.Rumball - Hinxworth, UK - April 2020

"If I have seen further, it is by standing on the shoulders of giants."
- Isaac Newton

"We in computer software insist on stepping on the toes of those who came before us instead of climbing on their shoulders".
- Dan Ingalls

“Myopia is still a problem even where there are giants' shoulders to stand on”
- Alan C. Kay

Section 1 Introduction

A long time ago in a galaxy really quite close by I designed a single board computer called the Microbox 2 (MB2) which was based around the 6809 and FLEX OS. It had a number of advanced (for 1982) features such as integrated EPROMdisk and RAMdisk, hi-res hardware accelerated graphics with a bitmapped text display that could use different languages and character sets (even Arabic), and a battery backed RTC and PRAM. As it had FLEX compatible drivers in EPROM, it could boot from any configured or un-configured copy of FLEX. It sold quite well, (I believe a few hundred were shipped worldwide), and launched my design career.

Not quite so long ago (2005), I revisited the MB2 design with a version built around a Xilinx 'Spartan' FPGA dev kit and although this worked well I wasn't happy with the design as it only kept to the spirit of the MB2 and couldn't run much of the original's software. Also due to a lack of space in the FPGA it wasn't possible to emulate the uPD7220A Graphics Display Controller (GDC) which was one of the defining features of the MB2.

Recent events have given me time to look once again at an updated version of the MB2 and the Microbox 2020 (MB2K2) is the result. This time there is a combination of custom hardware and software that aims be a complete emulation of the MB2 to the point of running the original's software without modifications.

The MB2K2 is a hardware based emulator built around an Xmos XU216 SoC where each of the 16 RISC cores in the XU216 map onto one of the individual LSI chips in the system being emulated, CPU, PIA, ACIA, GDC etc. The XU216 has 512KB of internal RAM and boots from an external QSPI serial flash device that holds the Xmos firmware and the promdisk for the 6809. As well as the XU216 the PCB includes a battery backed RTC, LEDs and DIP switches, VGA based video out, PS/2 keyboard interface and a USB connector for power and dual high speed serial ports which appear as CDC class virtual com ports to a Windows, MacOS or Linux based host computer.

The PCB has an abundance of test points and features to aid firmware development and is designed with an eye to easy manual assembly.

The MB2K2 is highly customisable. Xmos supply a free Eclipse toolchain that includes an IDE, compilers and debuggers that work with a low cost USB to JTAG debug interface allowing the user to change the MB2K2's firmware. Although initially the MB2K2 has been used to emulate the MB2, there's nothing to stop the emulation of other systems, processors and LSI devices.

The initial firmware release supports:-

- MC6809 processor emulation running at approx 8MHz equiv
- 64KB of 6809 RAM with MON09 and OS-9 L1 in 'ROM'
- 200KB ramdisk + 128KB non-volatile ramdisk & 3MB promdisk
- WD2123 DUART emulation with twin serial ports to the host computer as VCPs via USB, one can be used with FLEXNet for remote storage on FLEX and OS-9
- MC146818 RTC emulation via a physical battery backed RTC/PRAM on the PCB
- MC6821 PIA emulation for the MB2's bell and option switches and PS/2 keyboard interface (replaces the MB2's parallel keyboard
- uPD7220A hardware graphics accelerator emulation, 768x576 'VGA'
- PS/2 keyboard interface
- 80x80mm double sided 4 layer PCB which is designed to be assembled by hand with few fine pitch components and 'large' (0805) discrete throughout etc.

Changelist

Ver 0.90 - 2020-04-30 - Initial pre-release.
Ver 0.91 - 2020-07-30 - Add changes for OS-9 support.
Ver 0.95 - 2023-05-10 - PCB converted to KiCAD, add F-RAM based RAMdisk.

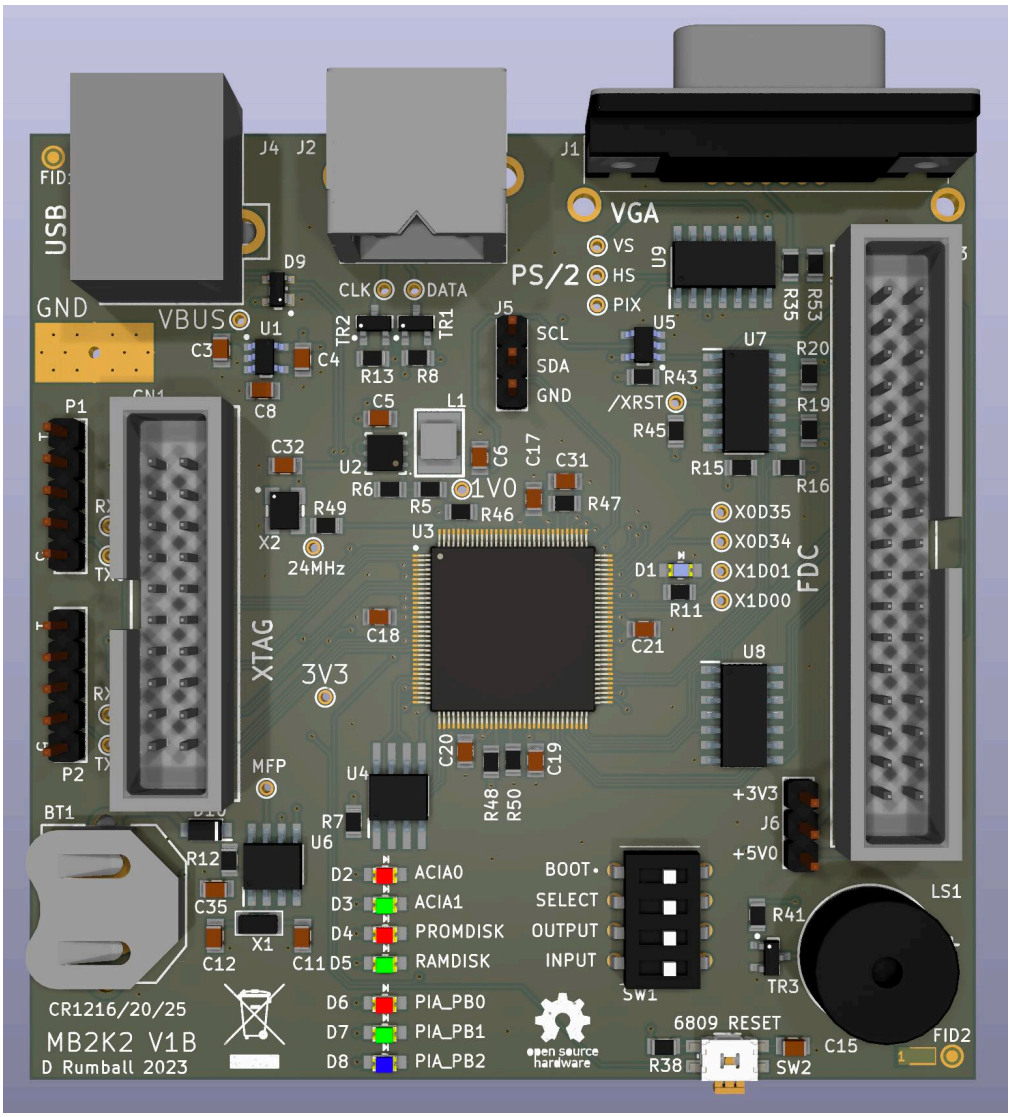
Known Issues

Todo List

This early release of the MB2K2 firmware is fully functional and runs the MB2 graphics demo software, but there are some unfinished parts and in later releases I hope to :-

- Improve the uPD7220A emulation for greater compatibility with MB2 software.
- Implement the WD1770 floppy disk controller

Section 2 - Hardware notes

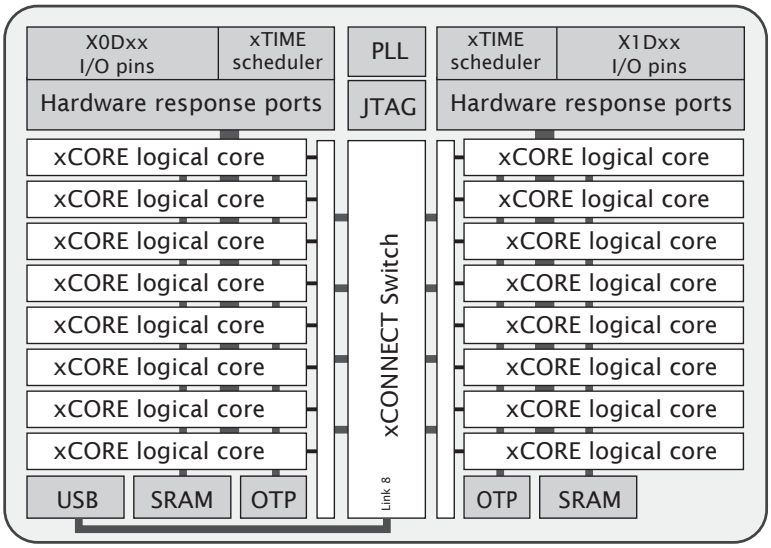


Like the original Microbox 2, the MB2K2 is based around a single PCB that carries the circuitry for the entire system. As well as the Xmos SoC the PCB includes :-

- 1V and 3.3V power supplies
- Power sequencing and reset circuity
- 24MHz system clock from which all timing is derived
- 8M bit QSPI flash device for firmware and PROMdisk
- Buffering for the VGA, floppy disk and PS/2 interfaces
- Battery backed RTC including persistent RAM/EEPROM
- 4 way DIP switch used for MON09 settings
- LEDs that show drive and serial port activity
- Sounder
- ‘Soft’ reset switch for the emulated processor
- Connectors for JTAG debug, USB, PS/2 keyboard, Floppy disk and VGA
- Multiple test points and test connectors for ease of development

A few concessions to the 21st century have been made due to the scarcity of monitors supporting analogue video, host computers with RS-232 serial ports and parallel 5V logic level keyboards. So the 15KHz interlaced video out of the MB2 has been replaced by a VGA compatible output, the serial ports are now CDC class virtual com ports via USB and the keyboard interface supports the ubiquitous PS/2 standard.

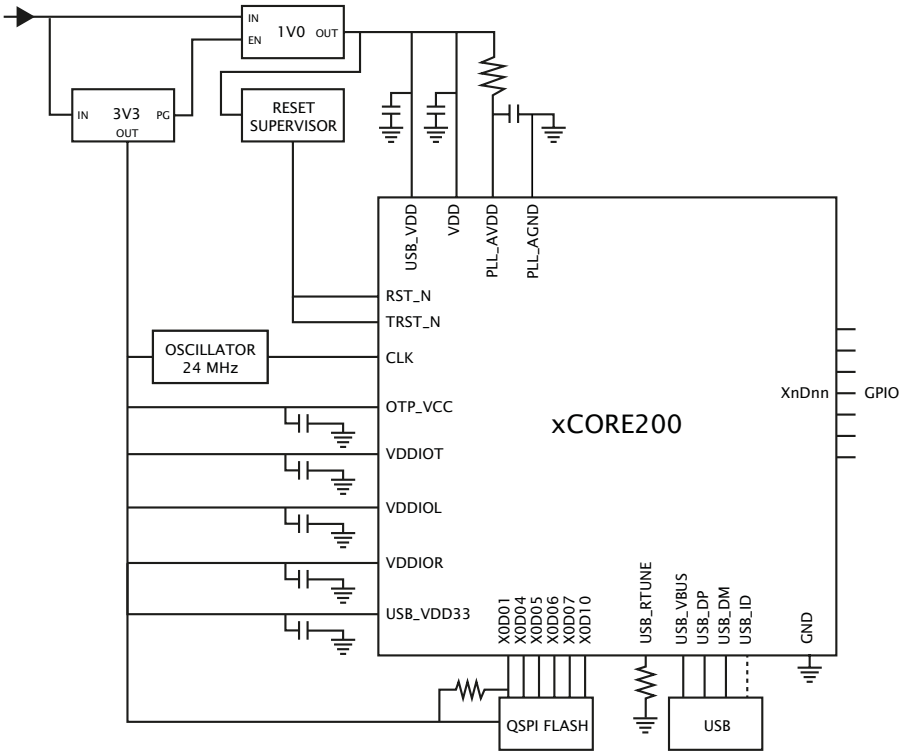
2.2 - XU216 SoC



The MB2K2 is based around an XU216-512 SoC from Xmos (part of the xCORE200 series) which has all of the processing and RAM required for the emulated system. Unlike conventional microcontrollers the XU216 runs multiple realtime tasks simultaneously and communicates between these tasks using an internal high speed network. xCORE microcontrollers are completely deterministic, so it's possible to code in software hard realtime functions that traditionally require dedicated hardware such as USB or video interfaces.

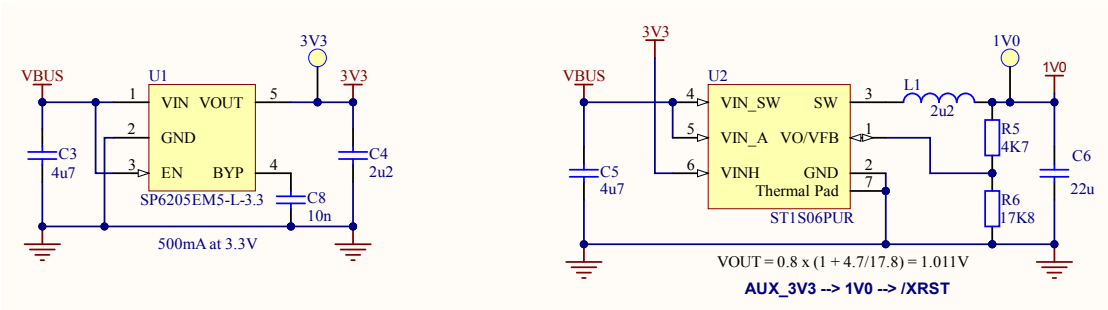
Key features of the XU216-512-TQ128 include:-

- Two tiles containing 8 32-bit 125MHz RISC cores with highly integrated I/O and on-chip memory
- A hardware based scheduler which performs functions similar to an RTOS. It services and synchronises events in a core and triggers cores on events generated by hardware resources such as the I/O pins, communication channels and timers. Once triggered, a core runs independently and concurrently to the other cores.
- Channels and channel ends allowing tasks running on cores to communicate using channels formed between two channel ends.
- A switch and links between tiles allowing channel communications to be routed. One channel travels off chip to the XTAG connector allowing real time debugging of running tasks.
- I/O pins connected to the processing cores by hardware response ports. The port logic can drive pins or sample the value on a pin optionally waiting for a particular condition.
- A set of programmable clock blocks that can be used to govern the rate at which ports execute.
- Each tile integrates a bank of SRAM for instructions and data, and a block of one-time programmable (OTP) memory that can be configured for system wide security features.
- A PLL used to create the core's processor clock given a low frequency external oscillator.
- A USB PHY providing High-Speed and Full-Speed, device, host, and on-the-go functionality.



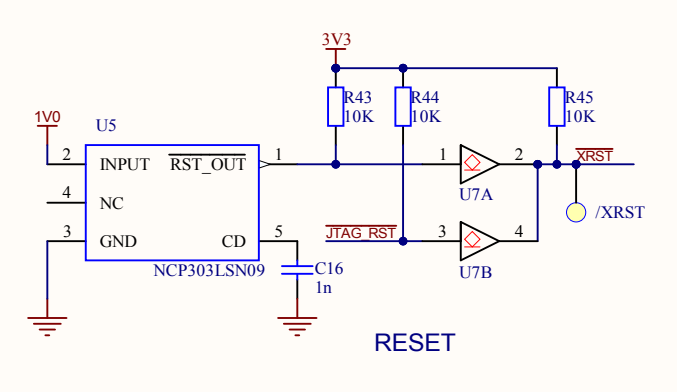
The MB2K2 is a simple implementation of the standard Xmos reference design for the XU216 only adding an external battery backed RTC, buffering or level shifting where necessary for the external ports together with option switches and indicator LEDs.

2.2 - Power



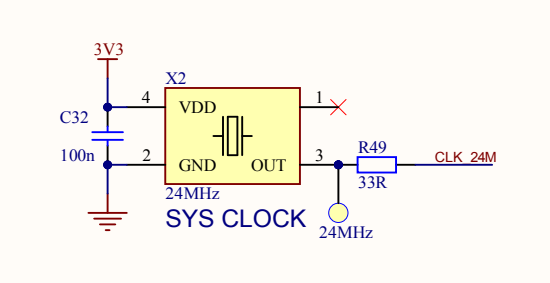
The MB2K2 is powered entirely via the USB connector and requires a notional 5V supply at a maximum of 250mA (approx 150mA nominal). The 5V supply powers the VGA video buffer and is passed to the PS/2 connector for the external keyboard. Two further power rails are derived from the 5V supply, 3.3V (via a linear regulator) which is used for I/O on the XU216, VDD for the QSPI flash, FDC buffers and the RTC and 1.0V (via a switch mode buck regulator) which is used for the XU216 core logic. The 1V rail is enabled by the 3.3V rail thus ensuring that the correct power sequence of 3.3V rising before 1V that is required by the XU216 is followed.

2.3 - Reset



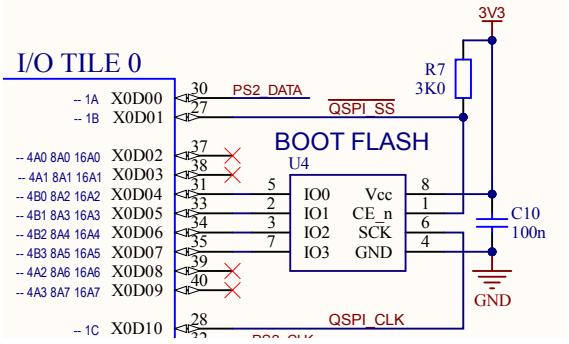
The Xu216 is reset on power up by a NCP303 series voltage detector which releases reset when the 1V rail has stabilised. A separate tact switch at the edge of the PCB is connected to the XU216 I/O and can be used as a ‘soft’ reset for the emulated processor.

2.4 - Clocks



All the clocks used in the MB2K2 (with the exception of the 32,768 Hz RTC clock) are derived from a single 24 MHz clock oscillator. Internally to the XU216 this 24 MHz clock is multiplied by a PLL to 48 MHz for the USB PHY, 125 MHz for the core logic and 31.25 MHz for the VGA pixel clock.

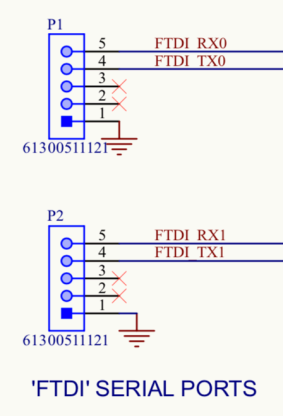
2.5 - Boot Flash



An 8-32 Mbit QSPI flash device is used to store the firmware for the XU216 and optionally in a separate data portion the PROMdisk for the emulated processor. This flash device is initially programmed by the Xmos toolchain via an ‘XTAG’ debug interface plugged into the PCB but may

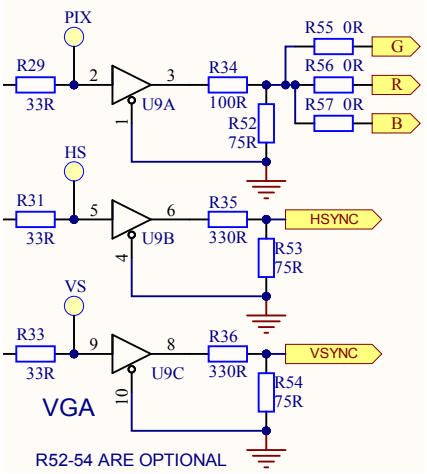
subsequently be read and written to programmatically by the running firmware.

2.6 - Optional serial ports (‘FTDI’)



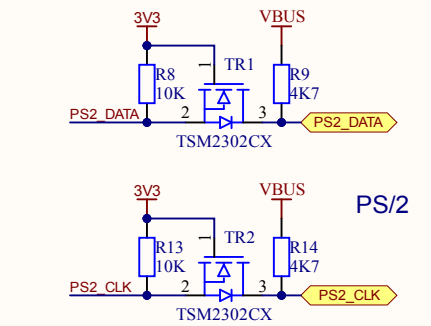
The MB2K2 design has connectors for two optional 3V logic level serial ports that are designed to connect to USB/serial cables such as the FTDI TTL-232R-3V3. Note that only Tx/Rx and GND are connected and care should be taken to ensure that any adapter cable uses 3V logic levels to avoid damage to the MB2K2.

2.7 - VGA interface



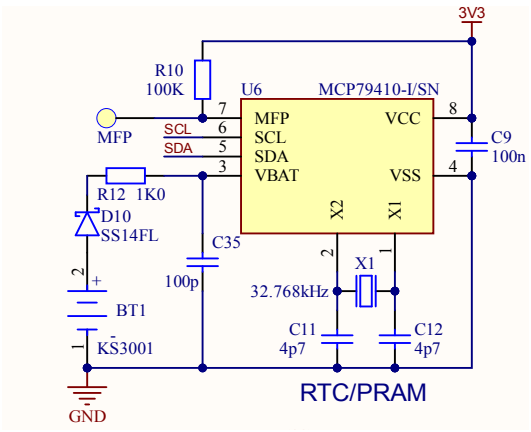
The VGA pixel and timing logic is contained purely in the XU216 and so only a simple buffer is required to drive the VGA 75Ω RGB and sync lines. Each output has a series current limiting R and an option termination R which is not required in most situations. The monochrome pixel output drives the R,G, and B lines equally for a White on Black display however removing one or more of the 0Ω links will allow other colours to be chosen such as Yellow or Green for a ‘retro’ phosphor CRT display look. :-)

2.8 - PS/2 keyboard interface



To replace the 'hard to obtain' parallel TTL keyboard of the MB2 the MB2K2 has an interface compatible with most PS/2 PC style standard keyboards. There are a pair of bi directional level shifters used to convert the 5V logic levels of the keyboard to the 3V logic levels of the XU216.

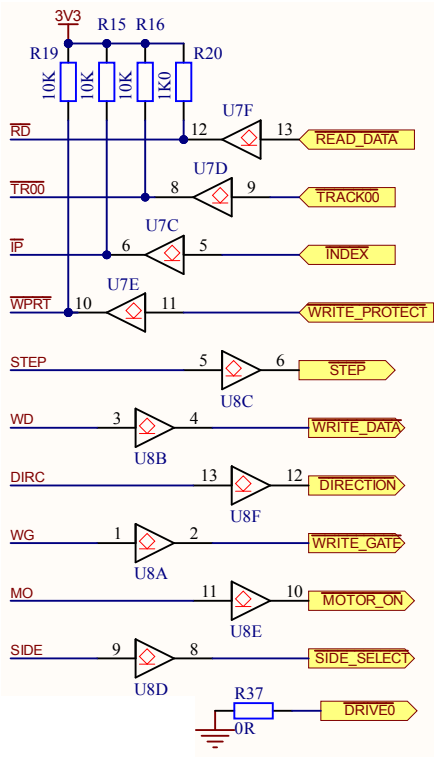
2.9 - Real Time Clock



The one significant piece of logic which cannot fit into the XU216 due to standby power concerns is the real time clock (RTC). This is implemented on the MB2K2 using a common MCP79410 variant from Microchip. This device includes a low power 64 byte SRAM and a 128 byte EEPROM that retains data even when the battery is disconnected. Timing for the RTC is provided by an internal oscillator driving a 32,768Hz crystal.The RTC is powered by either the 3.3V supply during operation or a PCB mounted 3V Lithium coin cell when the USB power is disconnected. The coin cell holder will accept either a CR1261, CR1220 or CR1225 cell. During standby the RTC clock keeps time and the contents of the SRAM are preserved.

As in the MB2, the RTC is used to automatically set the OS date (both for FLEX and OS-9) and the SRAM stores system parameters such as the allocation of logical to physical drive types and FLEX's ASN and TTYSET values. The MFP output of the RTC can used for calibration of the crystal oscillator for more accurate timekeeping.

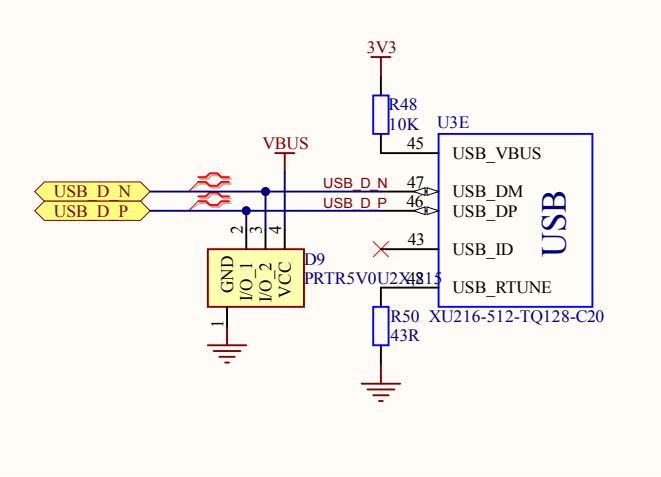
2.10 - Floppy disk interface



The MB2K2 supports the connection of a single 3/12” floppy drive with a standard 34 way .1” header and a pin out that matches the MB2. Simple open collector buffers are used that allow translation between the 3V logic levels of the XU216 and 5V logic levels of the floppy drive.

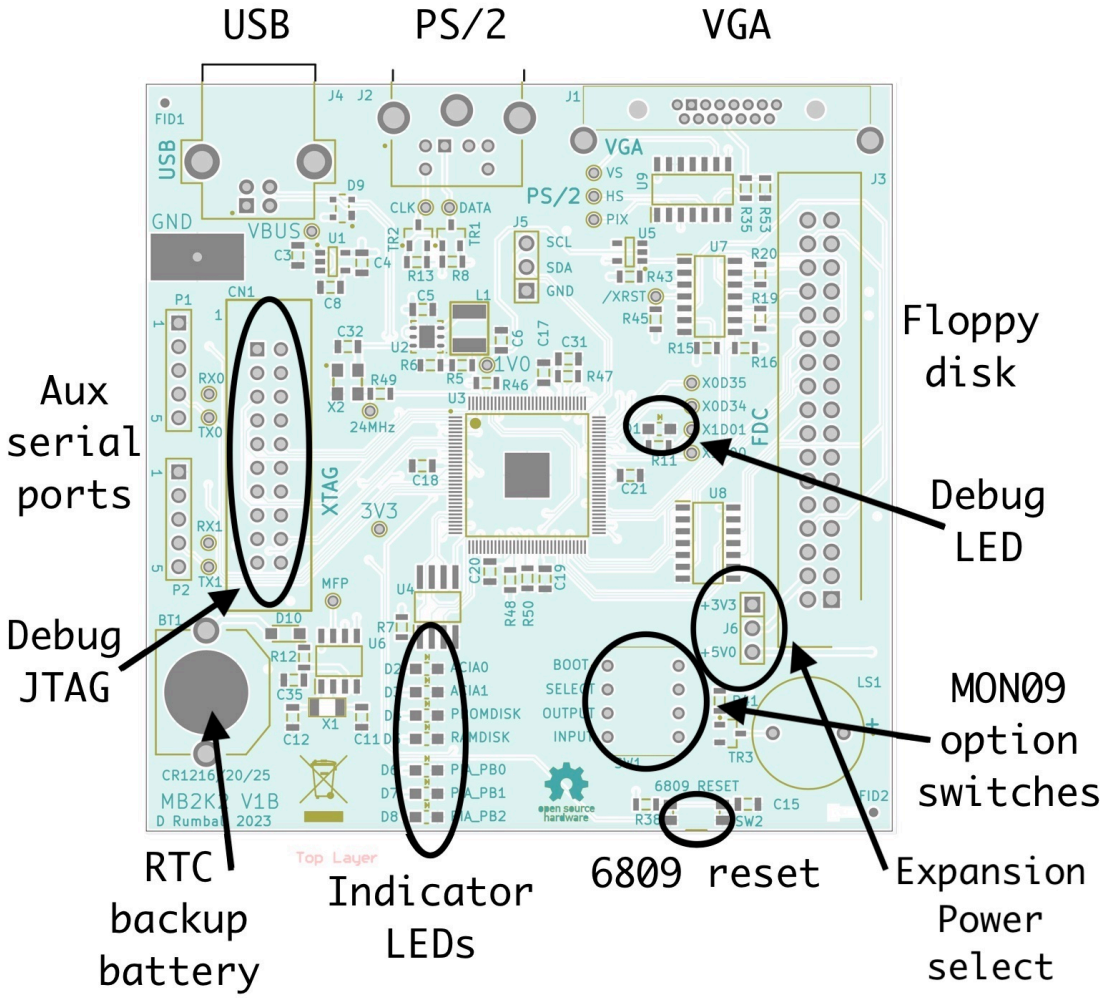
Note that the floppy disk interface emulation (WD1770) of the MB2K2 is not yet complete and will be included in a later release.

2.11 - USB interface

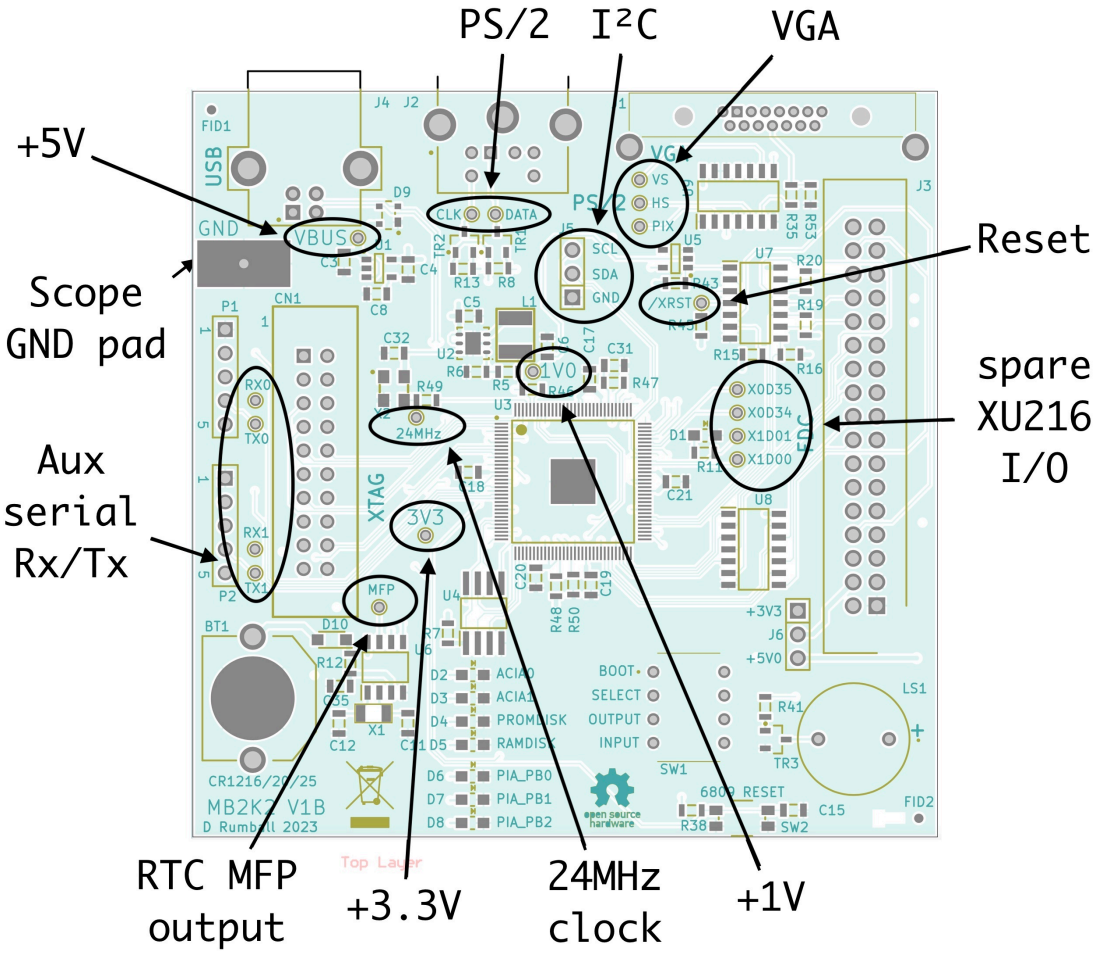


As the XU216 contains the USB PHY and all support logic, the USB interface hardware of the MB2K2 consists only of ESD protection.

2.12 - Connector, switches and LED locations



2.13 - Test point locations



Section 3 - Firmware notes

As the MB2K2 hardware is fairly simple and generic, the emulation of the MB2 or other systems is defined entirely by the firmware running on the XU216. However the resulting code looks quite different from a ‘pure’ PC based software emulation and indeed has a hardware like architecture that looks very different from usual.

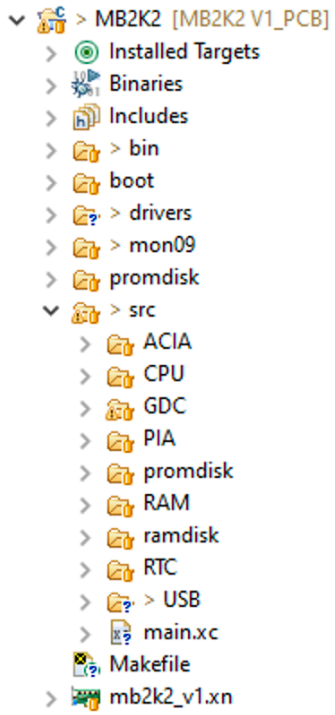
The MB2K2 project grew from the thought that the RISC processor cores and connecting hardware of the xCORE SoCs could map quite elegantly onto the LSI devices and busses of the MB2 where each core would emulate the internal operation of each LSI chip of the original and the channels connecting each core would map onto the address/data busses.

I’d recommend that the reader have a basic understanding of the xCORE architecture and XC language before reading this section. The firmware documentation of this release includes copies of the Xmos programming guide and XTIMEcomposer IDE/toolchain user guide to help with this.

The experienced Xmos developer might wonder why I haven’t used some of the more modern abstract features of the XC language such as combinable and distributable processes and interfaces which would potentially make the code more compact. This was a purely an aesthetic decision on my part as these higher level abstractions would blur the boundaries between the discrete mapping mentioned above and I really wanted to keep that.

3.1 - Project structure

The source file structure consists of a number of Xmos lib projects together with the MB2K2 project folder :-



- boot image and build tools
- drivers for USB CDC class for Windows7
- MON09 source code and build tools
- promdisk image DSK (used by xFLASH)
- serial port source
- 6809 source
- uPD7220 source
- 6821 and PS/2
- PROMdisk source
- decode/RAM handler source
- RAMdisk source
- RTC and I2C interface source
- XUD USB interface source
- hardware description file

There are two classes of build configurations in the project, ‘usb’ which uses the USB connection for serial connections and ‘ftdi’ which replaces the USB interface with a pair of buffered UARTs. Each is further split into ‘debug’ which has optimisation set to -O0 and adds support for the use of debugPrintf() and ‘release’ which has optimisation set to -O3 and removes all debugging support.

The release configs should be used when not debugging as the compiled code runs 3-5 times faster than the debug configs!

3.2 - Architecture and plumbing

Below is the ‘par’ statement that shows the placement of tasks on the tiles/cores and the channel connections between them (the ‘plumbing’).

```
par {
    on tile[0]: cpu_execute(c_addrData);                // (6809)
    on tile[0]: decodeMem(c_addrData, c_acia, c_rtc,    // (MC6883)
                          c_pia, c_ramdisk, c_promdisk,
                          c_gdc);

    on tile[0]: rtc(c_rtc);                             // (MC146818) I2C -> external RTC/EEPROM
    on tile[0]: pia(c_pia);                             // (MC6821) includes PS/2 keyboard interface
    on tile[0]: promdisk(c_promdisk);                   // external QSPI flash (1/2MB -> 3MB)
    on tile[0]: gdc(c_disp, c_gdc);                     // (uPD7220A) graphics engine
    on tile[0]: gdcDisplay(c_disp);                     // (uPD7220A) display interface
    //on tile[0]: fdc(c_fdc);                           // (WD1770)


    on tile[1]: DasBlinkenLights(p_debug_led);
    on tile[1]: ramdisk(c_ramdisk);                   // 40track DS/SD emulation (200KB)
    on tile[1]: acia(cdc_data[0], cdc_data[1], c_acia); // (WD2123)

    // USB interface
    on tile[1]: xud(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN, null, XUD_SPEED_HS, XUD_PWR_SELF);
    on tile[1]: Endpoint0(c_ep_out[0], c_ep_in[0]);
    on tile[1]: CdcEndpointsHandler(c_ep_in[CDC_NOTIFICATION_EP_NUM1],
                                    c_ep_out[CDC_DATA_RX_EP_NUM1],
                                    c_ep_in[CDC_DATA_TX_EP_NUM1],
                                    cdc_data[0]);
    on tile[1]: CdcEndpointsHandler(c_ep_in[CDC_NOTIFICATION_EP_NUM2],
                                    c_ep_out[CDC_DATA_RX_EP_NUM2],
                                    c_ep_in[CDC_DATA_TX_EP_NUM2],
                                    cdc_data[1]);
}// of par
```

The tasks are distributed between the tiles to balance memory usage and MIPs.

Tile 1 is defined as having the USB interface and this forces XUD, endpoint0 and the two CDC handlers plus the ACIA code to be on that tile as they share memory via interfaces. This also limits the number of tasks so as not to starve the hard real time task XUD of MIPs. Since the RAM usage of tile 1 will be low the RAMdisk task (which uses 200KB of memory) is placed here also. Finally the ‘blinky’ debug LED task is also placed on tile. Note that the compiler will warn that more cores than the limit of six for a USB tile are allocated but this is fine as the debug LED task uses minimal RAM/MIPs.

Tile 0 holds the remaining tasks with on core free for the future floppy disk controller interface emulation. Note that the uPD7220 emulation is split between two cores, one for the display interface that generates the hard real time VGA pixel and sync signals and a second for the drawing engine. These two tasks must be on the same tile as they share access to the approx 50KB frame buffer between them. The remaining RAM is used by the decode/mem task (64KB emulated RAM and 4KB shadow RAM) and code space for the remaining tasks.

The firmware can be customised with the free Xmos ‘xTIMEcomposer’ toolchain. Details on this are contained in the ‘Getting Started’ and ‘Firmware build & flashing’ documents included with this release.

3.3 - cpu_execute()

The 6809 code comes from an open source emulation by Arto Salmi, Joze Fabcic and Brian Dominy. This was chosen more or less at random and most likely is not the fastest or most compact example. However, it does work! Note that this is an implementation in ‘C’ rather than ‘XC” and as such needs to use an external library to support channels and other XC features. This lib is called ‘xcore-c’ and features support for channel and streaming channels, ports and clock blocks, timers, hardware locks , select and interrupt events.

The CPU communicates via a channel with the decodeMem task. For a read transaction, the CPU’s reset switch state is read from bit 31 of the word and used to reset the CPU if asserted.

3.4 - decodeMem()

The decodeMem task emulates the 6883 ‘SAM’ chip of the MB2 and as such controls access to the CPU’s 64KB address space. In this implementation it also decodes addresses from the CPU and handles access to the peripheral devices such as the PIA and DUART (ACIA) via channels to the tasks emulating those devices.

A simple transaction protocol is used where the 32 bit word transferred from the CPU has the address in the low 16 bits, data in the next 8 bits and a transaction type flag in the upper 8 bits. This flag is 0x00 for a read cycle and 0xFF for write. This data is then written to RAM or if the access is in a peripheral address range, the word is passed to the peripheral. For a read transaction, the byte read from RAM or the peripheral device is sent back along the same channel in the lowest 8 bits.

For CPU read transactions, the decodeMem task reads the ‘soft’ CPU reset switch and if the switch input is low (switch pressed) the task debounces the input and waits for the switch to be released then sends the reset to the CPU by setting bit 31 of the returned word.

The CPU memory is reset after power up and two sections are initialised, the first contains an image of the OS-9 kernel and boot modules from $0000-$2FFF and the second contains the MON09 ‘eprom’ image from $E000-$FFFF. This initial data is added to the source by means of a header file. This file is generated by a batch command that should be run after the MON09 or os9 6809 sources are modified. Details of this procedure may be found in section 4.

In the same manner as the MB2 a 4KB section of RAM at $E000 may be remapped from ‘eprom’ to RAM. This area is called the ‘shadow RAM’ and is mapped in whenever control is passed from MON09 to FLEX. In the MB2, this area was used by several programs and FLEX utilities so is emulated in the MB2K2. A portion of the shadow RAM is initialised after reset in the same way as the main RAM. In this case a bit mapped character set is placed from $E400 up and this is used by some versions of the MON09 ‘GDCOUT’ character drawing routine. See the GDC description for more details. As for the main RAM, this initialisation is defined by a header file more details of which may be found in section 4.

MB2K2 address map

```
-- RAM          $0000 - $AFFF ($0000 - $BFFF for FLEX)

-- OS-9 modules $B000 - $DDFF

-- MON09
        $E000 - $EFFF monitor commands (switched out when FLEX running)
        $F000 - $FFFF monitor subroutines and drivers
        $DE00      Scratch RAM + stack space.
        (RAM+127-16) Top of system stack.
        (RAM+384)  Start of scratch space.

-- IO space
        $FF00      I/O base address.

        PIA1
        KEYREG   EQU   $FF00
        PIACA    EQU   $FF01
        SYSREG   EQU   $FF02
        PIACB    EQU   $FF03

        DUART
        UARTD1   EQU   $FF08
        UARTC1   EQU   $FF09
        UARTD2   EQU   $FF04
        UARTC2   EQU   $FF05
        BAUD1    EQU   $FF0C
        BAUD2    EQU   $FF0D

        FDC
        COMREG   EQU   $FF10
        TRKREG   EQU   $FF11
        SECREG   EQU   $FF12
        DATREG   EQU   $FF13

        GDC
        GDCPRM   EQU   $FF14
        GDCCOM   EQU   $FF15

        RTC
        RTCADD   EQU   $FF18
        RTCDAT   EQU   $FF19

        RDC (ramdisk controller)
        COMREG   EQU   $FF20
        TRKREG   EQU   $FF21
        SECREG   EQU   $FF22
        DATREG   EQU   $FF23
        MODREG   EQU   $FF24

        PDC (promdisk controller)
        COMREG   EQU   $FF30
        TRKREG   EQU   $FF31
        SECREG   EQU   $FF32
        DATREG   EQU   $FF33
        MODREG   EQU   $FF34
```

```
FRDC (f-ramdisk controller)
COMREG   EQU   $FF58
TRKREG   EQU   $FF59
SECREG   EQU   $FF5A
DATREG   EQU   $FF5B
MODREG   EQU   $FF5C
```

3.5 - rtc()

The RTC task emulates the 146818 RTC used on the MB2. The code passes RTC register accesses to an external Microchip MCP7941 series device via a simple 'bit banging' I²C interface. The emulation involves register address remapping, transposition from binary to BCD formats and spoofing of power fail detection.

One thing to note is that there is a calibration value defined in the code which allows adjustment of the RTC frequency as per section 5.2.3 of the MCP7941X data sheet.

```
#define CALIBRATION_VALUE 0x7D // + 250 32.768KHz clocks/min
```

This value should be calculated from the time drift measured over a period of days for each individual system for increased RTC accuracy.

In the current implementation the RTC process also handles the F-RAM accesses. This is due to the fact that the F-RAM is connected via I2C and the RTC process 'owns' the GPIO bits used for SCL and SDA.

3.6 - pia()

The PIA task emulates the 6821 PIA used on the MB2 for keyboard input on port A and the system register on port B that had the following bit mappings:-

```
PIA port B definitions (SYSREG)
 * bit 0 - 3 are outputs
 * 0 - DRV     (DRV and /DDEN are used by the floppy disk interface)
 * 1 - /DDEN
 * 2 - MAP bit (maps out bottom 4K of monitor ($E000-$EFFF) when 0)
 * 3 - BELL
 * bit 4 - 7 are inputs
 * 4 - Initial input port
 * 5 - Initial output port
 * 6 - Select OS to auto boot
 * 7 - Auto boot OS
```

The MB2K2 has a 4 way DIP switch to to match the MB2 and the switches have the following functions:-

```
Switch 0   Sets the auto boot function.
                    on  = Auto boot into FLEX or OS-9 on reset
                    off = run MON09 on reset

Switch 1   Selects the OS to boot into.
                    on  = Auto boot into OS-9 on reset
```

```
                    off = Auto boot into FLEX on reset

    Switch 2   Sets the initial output port to be used after reset
                    on  = PORT 1 (serial port 0) - USB
                    off = PORT 0 (GDC screen)

    Switch 3   Sets the initial input port to be used after reset
                    on  = PORT 1 (serial port 0) - USB
                    off = PORT 0 (PS/2 keyboard)
```

As it is now difficult to obtain parallel TTL keyboards this is emulated by including a PS/2 compatible keyboard interface in the PIA task. Keypress messages from the keyboard are mapped to ASCII values by tracking the key up/key down and shift states and applying these states and the raw key value to a pre defined look up table. This table is composed of four sections, one each for shift and shift lock on/off. This mapping is defined in the keycap header file and this file can be simply edited to change the key mapping as detailed in section 4. The default mapping is included in appendix 3 and is defined to be compatible with the 'CEDRIC' screen text editor which is included as part of the standard PROMdisk.

3.7 - promdisk()

The PROMdisk task functionally emulates the EPROMdisk of the MB2 but doesn't directly emulate the 8255 PIA in the original design. Instead this task acts as an interface to the external QSPI flash that is used as the boot image for the XU216. This flash is split into two partitions, the first holds one or more boot images and the second holds a combigned disk image (.DSK) file containing the MB2K2's system volume for FLEX followed by the boot volume for OS-9. Typically for a 4MB flash device the first 1MB is dedicated to the XU216 boot images and the remainder for the PROMdisk.

The promdisk image is loaded into the data portion as part of the firmware flashing process as mentioned in section 4 of this guide and the separate 'Firmware build and flashing' guide.

The format of each PROMdisk defaults to a 192 track single sided/ single density disk of approx 1.44MB.

```
#define SECTORS_PER_TRACK  30
#define TRACKS_PER_DISK    192
#define BYTES_PER_SECTOR   256
#define PROMDISK_SIZE      1474560
```

Note that this is not one of the standard FLEX formats and will not be compatible with certain FLEX diagnostic utilities that require a drive to be one of a limited number of standard formats.

Other formats may be used for user defined PROMdisks as the firmware reads the System Information Record (SIR) of the FLEX disk image in flash and uses this to define the correct offsets into the image. The combigned FLEX/OS-9 PROMdisk may be any size up to 3MB.

Under FLEX the PROMdisk is by default read only and any attempt to write to the disk will return an error. However there is a flag in MON09's memory that can be set to allow writes to the PROMdisk by FLEX and this flag

is set by the ‘PDRW’ (Promdisk read/Write) command and reset by ‘PDRO’ (PROMdisk read only). A reset will set the default to read only.

Under OS-9 the PROMdisk is set to r/w always.

Note that there is no wear levelling and repeated writes (10,000's) will eventually 'wear out' the part of the flash image containing frequently written sectors.

3.8 - gdcDisplay()

The GDC emulation consists of two separate tasks, the first (gdcDisplay()) converts the the graphics and text frame store data to a VGA compatible serial data stream and sync signals whilst the second interprets commands from the 6809 and emulates the drawing engine of the 7220A.

The display timings are all derived from the XU216’s 125MHz core logic clock. This is divided by four to give a pixel clock of 31.25MHz which then drives the output ports for the pixel data and sync signals.

```
// outputs are all clocked at pixel rate
// pixel clock = 31.25MHz (1/4 of 125MHz system clock)
   configure_clock_rate(pixel_clk, 125, 4);
   configure_out_port(p_pix, pixel_clk, 0);
   configure_out_port(p_hs,  pixel_clk, 1);
   configure_out_port(p_vs,  pixel_clk, 1);
   start_clock(pixel_clk);
```

The ports are configured to accept 32 data writes which are then buffered and serialised using the pixel clock.

```
on tile[0] : buffered out port:32 p_pix = XS1_PORT_1M;
on tile[0] : buffered out port:32 p_hs  = XS1_PORT_1N;
on tile[0] : out port p_vs  = XS1_PORT_1O;
```

In this way it’s possible to run a real time display with a simple code loop and the hardware task switching inherent in the XU216 architecture ensures that the timing of the video outputs cannot be disturbed by the other tasks running on the device.

The code to generate a line of video then is just a few simple statements :-

Note that the bit order of the 32 bit word needs to be reversed and the two 16 bit sub words swapped so as to output bits in the correct order for display.

As per the 7220A two display partitions are supported and the starting addresses and lengths of these portions are passed to the display task from the drawing engine task aver a channel which is synchronised to the vertical blanking intervals so that partition changes do not disturb the display.

Because the VGA output is non interlaced there is no need for the interlaced/non interlaced display switching on the MB2. Instead the text display mode simply emulates a ‘repeat field’ display by repeating each scan line to give the same effect. The repeat field parameter is passed into the

```
// partition 1
      if (lengthPart1 != 0) {

          h = startPart1;

          for(l=0; l < lengthPart1; l++) {

// horizontal front porch
              p_hs  <: 0xFFFFFFFF;
              p_pix <: 0x00000000;

// horizontal sync
              p_hs  <: 0x00000000;
              p_pix <: 0x00000000;

// horizontal back porch
              #pragma loop unroll
              for (i=0; i<HBP_WORDS; i++ ) {
                  p_hs  <: 0xFFFFFFFF;
                  p_pix <: 0x00000000;
              }

// active line
              #pragma loop unroll
              for (i=0; i<HAL_WORDS; i++ ) {
                  p_hs  <: 0xFFFFFFFF;
                  p_pix <: bitrev(((vram[h] & 0x0000FFFF) << 16) | ((vram[h] & 0xFFFF0000) >> 16)) & blank;
                  h++;
              }

// if repeat field at end of even lines step pointer back a line
              if ((repeatLine) && ((l/2)*2 == l)) {
                  h -= HAL_WORDS;
              }

          }// of active lines for

      }// of partition 1
```

display task along with a ‘display on/off’ parameter together with the partition information.

Note that the frame store memory is shared between the display and drawing engine tasks and unlike normal ‘C’ this is treated as an error by the XC compiler. This may be circumvented by using the ‘unsafe’ pragma that informs the compiler that we really do know what we’re doing and that it should allow memory sharing.

3.9 - gdc()

This is the second task of the GDC emulation which handles communication with the 6809 and implements the drawing engine of the 7220A. The emulation of the drawing engine is not complete as of the time of writing (ver 0.9) but it is sufficient to work with MON09’s graphics drivers and the majority of the MB2’s software including the large graphics demo. The missing parts include the general purpose ‘fill’, zoom, rectangle and circle drawing and any operation that reads data back from the GDC. These missing parts are planned to be implemented in later versions.

The GDC emulation also doesn’t have the command and parameter FIFOs of the 7220A, however this is a deliberate omission as the channel used to communicate between the decodeMem task and the GDC task inherently contains buffering and the blocking nature of the channel communication protocol means that the FIFOs are unnecessary. Likewise the ‘busy’ bit of the GDC’s stays register isn’t implemented as the channel will just harmlessly stall if the graphics engine is busy.

There is one significant difference between the emulation and the 7220A and that is the addition of a new command and corresponding sub commands not in the original chip. This command is designed to act as an

accelerator for various ‘bottleneck’ routines such as cursor location generation and graphics character drawing.

The base command is 0xDx and the sub commands are listed below :-

```
// command is of form Dx where x is the four bit sub command :-
// x = 0 - draw char at existing eAD/dAD
//     1 - draw inverse char at existing eAD/dAD
//     2 - process char in terminal emulator using embedded char set *
//     3 - cursor off
//     4 - cursor on
//     5 - toggle cursor
//     6 - calculate eAD/dAD from row, col and offset
//     7 - calculate eAD/dAD from x,y (SETCRG replacement)
```

Sub commands 0 and 1 are faster versions of the 7220A’s ‘GCHRD’ that use the internal char set and assume a fixed character cell size and drawing direction while sub command 2 implements a full terminal emulator that just takes a stream of ASCCI characters and control codes. The control codes match those of the terminal emulator in MON09 and are listed in appendix 1. Before using this internal terminal emulator it should be initialised by sending it a 0x0C (CLEARSCREEN) character.

Note that the terminal emulator character set is defined in char_set.h where after an initial eight bytes that describe the character set attributes each character is represented by an 11 byte byte sequence thus :-

```
    // character set attributes
    84, // screen chars/row
    24, // screen rows
    7,  // char x pixels
    11, // char y rows
    9,  // bounding box pixels
    12, // bounding box rows
    4,  // initial attributes
    11  // char spacing (pixels)
    .
    .
    .
    , 0b00111000 //Character = '&'
    , 0b01000100
    , 0b01000100
    , 0b00101000
    , 0b00010000
    , 0b00101001
    , 0b01000110
    , 0b01000110
    , 0b00111001
    , 0b00000000
    , 0b00000000
    .
    .
    .
```

The graphics display resolution is 768 1 bit pixels by 576 lines whilst the text display resolution is 768 1 bit pixels by 288 lines and the two frame stores exist as a contiguous section of RAM of (768*576/16)*3/2 words or approx 81KB. This is defined as a array of ‘short’ (16 bit) words to match the 7220A’s memory width. The base address of the graphics display

is 0x0000, the text display starts at 0x6C00 (16 bit words) and the top of the frame store is 0xA1FF (16 bit words).

The graphic frame store is loaded at startup with an image defined in 'bootImage.h'. This image can be viewed by using the 'graph' command after booting FLEX but before running any other graphics command.

3.10 - fdc()

The floppy disk controller will be implemented in a later release.

3.11 - DasBlinkenLights()

This task implements a simple 'breathing' LED that is used to indicated that the firmware is up and running. It uses a timer to trigger events and so uses the minimum of MIPs.

3.12 - ramdisk()

In the same manner as the PROMdisk the RAMdisk task is functionally equivalent to the RAMdisk of the MB2 but doesn't directly emulate the original design. In the MB2 the RAMdisk uses part of the GDC's memory but in the MB2K2 the RAMdisk memory is a separate part of the XU216's RAM.

The RAMdisk's format is defined in the header file:-

```
#define SECTORS_PER_TRACK  20
#define TRACKS_PER_DISK    40
#define BYTES_PER_SECTOR   256
#define RAMDISK_SIZE       204800
```

After reset the RAMdisk is formatted according to the defined format.

3.13 - F-RAMdisk

This is a new addition for the V1B PCBs and has no analogue to the original MB2 design. It is intended to free up internal Xmos RAM used by the existing RAMdisk thus creating space for larger processor emulation such as the 68K in future releases. The F-RAMdisk is non volatile and has no limitations on the number of writes (unlike flash) and so is ideal for a local scratch disk in cases where there is no remote storage connected via FlexNet.

Unlike the RAMdisk the F-RAMdisk isn't formatted after power up and so must be formatted before first use. For Flex, this can be achieved either by the MON09 'DF' command or the 'framdisk' utility. For OS-9 the usual 'format' command can be used. The F-RAM device has a capacity of 128KB.

3.14 - acia() and USB

The acia task emulates the 2123 DUART used on the MB2. Rather than an implementation of a standard UART however, the MB2K2's serial ports pass through the USB connection as CDC class virtual com ports. These ports are implemented as per the Xmos app note 'AN00184 USB CDC Class as Virtual Serial Port' and use a total of four cores on tile 1:-

- A core running the XUD USB device library.
- A core implementing Endpoint0 responding to both standard and CDC class-specific USB requests.
- Two cores handling the data endpoints and notification endpoints of the CDC ACM class, one for each virtual com port instance.

The acia task communicates with the endpoint handlers by means of an XC interface of type 'usb_cdc_interface' which has methods to check for received data (available_bytes) and read/write characters. (get_char & put.char). The four USB cores communicate in part using shared memory so need to be placed on the same tile. Note that the virtual com port doesn't have a defined baud rate associated with it at the device end so the monitor routines and storage associated with that are not implemented on the MB2K2.

It's recommended that Windows 10 is used for the host computer (if using Windows is required) as this natively supports dual CDC class devices, however if Windows 7 is used then a driver needs installing on the host system. This driver is included in the release and is located in the MB2K2 project folder. No driver is of course required for MacOS or Linux.

The firmware also supports replacing the USB interface with a pair of UARTs which are attached to the auxiliary serial connectors on the PCB. These are intended to be used with USB to serial conversion cables such as the FTDI 'TTL-232R-3V3' in situations where the USB connection is used for power only.

The UARTs are configured for baud rate of 115200 with eight bit data, no parity and one stop bit and have 1KB buffers on the receive side.

Section 4 - Customisation

In essence the entirety of the MB2K2 firmware is customisable as the full source code is available in the releases however within the existing MB2 emulation there are also sections that can be customised without changing the underlying firmware and these are detailed in this section.

4.1 - MON09

The source of the embedded 6809 assembly language MON09 monitor is located in the ‘mon09’ folder in the `mB2K2 project. After modifying the source it needs to be assembled and then merged into the 64KB memory array together with the FLEX image.

To assist with this process there is a batch file in the MON09 folder which will run the assembler (A09), merge the generated SREC file and FLEX SREC file into the memory array definition then copy the resultant header file into the source files. When the project is subsequently built the new version of MON09 will be included.

Included with the MON09 sources are optional versions of the GDC drivers that use the new additions to the GDC’s command set mentioned in the firmware description section.

GDCSUBS_V1.TXT - use the original unmodified GDC commands
GDCSUBS_V2.TXT - use the new versions of SETCRG, POINT and LINE (faster)
GDCOUTV1.TXT - use the original unmodified GDC commands for text display
GDCOUTV2.TXT - use the versions of RCTOCR and PUTCHAR (faster)
GDCOUTV3.TXT - use the new GDC internal terminal emulator (fastest)

4.2 - O/S images

The source code for the OS-9 kernel and core modules that are pre loaded into RAM is in the ‘os9’ folder in the project. After modifying any of these files it is necessary to run the ‘build’ script in the os9 folder followed by the ‘build’ script in the ‘MON09’ folder in order to merge the assembled files into the RAM image.

To use a different version of FLEX simply replace the ‘flex.cor’ file on the PROMdisk image (FLEXPD.dsk) with the new version and then run the ‘build’ script in the PROMdisk folder to merge the FLEX and OS-9 PROMdisk images into the single MB2K2PD.dsk image that is used by xFlash.

4.3 - GDCOUT character set

The character set used by MON09’s native terminal emulation (not the GDC’s internal version) is pre loaded into the 6809’s ‘shadow RAM’ by a header file in the RAM folder in the MB2K2 project. After the initial padding bytes there are eight bytes that describe the character set attributes and after that each character is represented by an 11 byte byte sequence thus :-

```
// character set attributes
    84, // screen chars/row
    24, // screen rows
    7,  // char x pixels
    11, // char y rows
    9,  // bounding box pixels
    12, // bounding box rows
    4,  // initial attributes
    11  // char spacing (pixels)
    .
    .
    .
    , 0b00111000 //Character = '&'
    , 0b01000100
    , 0b01000100
    , 0b00101000
    , 0b00010000
    , 0b00101001
    , 0b01000110
    , 0b01000110
    , 0b00111001
    , 0b00000000
    , 0b00000000
    .
    .
    .
```

4.4 - PS/2 keyboard mapping

Keypress messages from the PS/2 keyboard are mapped to ASCII values by tracking the key up/key down and shift states and applying these states and the raw key value to a pre defined look up table. This table is composed of four sections, one each for shift and shift lock on/off. This mapping is defined in the ‘keymap.h’ header file located in the PIA folder in the MB2K2 project and this file can be simply edited to change the key mapping. An example of the mapping is shown below.

```
  0x00      //Scancode = $00
 ,0x05      //Scancode = $01       F9    CEDRIC - delete char
 ,0x00      //Scancode = $02
 ,0x19      //Scancode = $03       F5    CEDRIC - cut
 ,0x0E      //Scancode = $04       F3    CEDRIC - search and replace
 ,0x03      //Scancode = $05       F1    CEDRIC - search forward
 ,0x1A      //Scancode = $06       F2    CEDRIC - search back
 ,0x08      //Scancode = $07       F12   CEDRIC - delete left (backspace)
 ,0x00      //Scancode = $08
 ,0x17      //Scancode = $09       F10   CEDRIC - delete word
 ,0x13      //Scancode = $0A       F8    CEDRIC -
 ,0x10      //Scancode = $0B       F6    CEDRIC - paste
 ,0x01      //Scancode = $0C       F4    CEDRIC - global replace
 ,0x09      //Scancode = $0D       Tab
 ,'`'       //Scancode = $0E       `
 ,0x00      //Scancode = $0F
 ,0x00      //Scancode = $10
 ,0x00      //Scancode = $11
 ,0x00      //Scancode = $12
 ,0x00      //Scancode = $13
 ,0x00      //Scancode = $14
 ,'q'       //Scancode = $15       q
```

```
 ,'1'      //Scancode = $16      1
 ,0x00     //Scancode = $17
 ,0x00     //Scancode = $18
 ,0x00     //Scancode = $19
 ,'z'      //Scancode = $1A      z
 ,'s'      //Scancode = $1B      s
 ,'a'      //Scancode = $1C      a
 ,'w'      //Scancode = $1D      w
 ,'2'      //Scancode = $1E      2
 ,0x00     //Scancode = $1F
 .
 .
 .
```

4.5 - RTC PRAM defaults

These are loaded if MON09 detects that power has been lost to the battery backed RTC and are defined in the MON09 file 'minit.txt' as the table 'RTCTAB' and the values here can be simply edited to change the defaults.

```
RTCTAB FCB %00001111 MB2 values, not used in the MB2K2.
 FCB $AA powerfail flag
 FCB 0,1,2,$FF promdisk, ramdisk, f-ramdisk, unassigned
 FCB $7F,0,$3A,$18,$50,0,0,$08,0,0,$1B TTYSET parameters.
 FCB 0,1 ASN parameters.
 FCB $00,$00,$00,$00,$00,$00,$00,$00 GDC timing parameters
 FCB 0,0,0,0,0,0,0 reserved
 FCB $00,$11,$22,$33,$44,$55,$66,$77,$88,$99,$AA,$BB,$CC,$DD,$EE,$FF user
params
```

4.6 - PROMdisk image

The PROMdisk .DSK disk image that is loaded into flash is stored in the 'promdisk' folder in the MB2K2 project. It is formed by running the 'build' script in the PROMdisk folder to merge the FLEX and OS-9 PROMdisk images into the single MB2K2PD.dsk image that is used by xFlash.

4.7 - boot image

The graphics frame store is initialised at startup with an image defined in the 'bootimage.h' header file in the GDC source folder. This image is formed of a byte array generated using the image2cpp utility at https://javl.github.io/image2cpp/

Section 5 - MON09 commands

There are fewer commands in the MB2K2 version of MON09 compared to earlier versions. This is because the majority of debugging can now be done directly at the OS level since FLEX is 'built in' to the basic system and so software such as the TSC Debug package can be used instead of the lower level debugger. The space freed up has been used for additional drivers instead of commands. Some basic commands have been left in however, and these are documented below.

There are twenty three monitor commands, each represented by a two letter name. Typing the two letters will invoke that command, which will then prompt for any necessary parameters. There are four types of parameter:-

```
Four digit hex number.................XXXX
Two digit hex number...................YY
One digit hex number.....................Z
Text string or character................T
```

All commands case insensitive.

5.1 - Memory commands

The first six commands are concerned with examining, modifying and testing memory. Of these, the first two have a common control format where a CR will examine the next location or page, a '-' will examine the previous location or page, and any other character will exit the command.

```
Command: DU   Dump memory
Format:  Hex and ASCII dump of memory from XXXX
Action:  Displays a 256 byte block of memory as two digit hex
         values and ASCII. Any non-printable character will represented
         by a '.'
```

```
Command: ME   Memory Examine and alter
Format:  Memory examine and alter from XXXX
Action:  Displays an address and the contents of that address. The
         contents may be changed by typing a space followed by the new
         two digit value. A verify is performed on the location changed.
```

```
Command: PO   Poke Memory
Format:  Poke memory location at XXXX value YY
Action:  Deposits the data into the location without verifying or
         reading the next address. Used for testing memory mapped
         peripheral devices where a read would corrupt data.
```

```
Command: PE   Peek Memory
Format:  Peek memory location at XXXX
Action:  Displays the data stored at location without reading the next
         address.
```

```
Command: TM   Test Memory
Format:  Test memory from XXXX to XXXX
Action:  Tests memory in the range given. Any data in the memory
         will be overwritten.
```

Note that 'TM' command is largely redundant in the MB2K2 as the RAM is internal to the XU216 and so extremely unlikely to have any errors, however this command has been kept as it provides a handy benchmark of CPU performance between different implementations.

```
Command: FM   Fill Memory with constant
Format:  Fill memory with constant from XXXX to XXXX value YY
Action:  Fills the indicated memory range with the data.
```

5.2 - I/O commands

The monitor input/output may come independently from one of three sources :-

PORT NUMBER	INPUT	OUTPUT
0	Keyboard	Internal display
1	serial port 0	Serial port 0
2	serial port 1	Serial port 1

The initial ports are set on reset by the configuration switches.

```
Command: SI   Set Input port
Format:  Set input port to Z
Action:  Sets the active input port
```

```
Command: SO   Set output port
Format:  Set output port to Z
Action:  Sets the active output port.
```

5.3 - Debug commands

The next five commands are concerned with running programs directly from within MON09. A breakpoint can be set with the BP command which when reached will return control to MON09 with an automatic display of registers. The register values may be modified using the ME command. (The register values are stored in the 10 bytes below the location pointed to by the stack pointer S.)

```
Command: DR   Display Registers
Format:  Display registers
Action:  Displays the current program register set.

Command: BR   Set breakpoint
Format:  Set Breakpoint at XXXX
Action:  Replaces the byte pointed to by a SWI ($3F). The breakpoint is
         cleared when hit and the saved byte returned.
```

```
Command: RP   Run Program
Format:  Run program from XXXX
Action:  Loads the processor registers, then jumps to program starting
         at address given.


Command: CP   Continue Program
Format:  Continue program after SWI....
Action:  Continues execution of a program from a breakpoint instruction.


Command: JU   Jump to program
Format:  Jump to program at XXXX
Action:  Execute a program starting at the given address without
         loading the registers first.
```

5.4 - Disk commands

There are five commands for disk control and testing. Note that any errors reported will be a copy of the emulated disc controller status register.

```
Command: RS   Read Sector
Format:  Read sector on drive Z track YY sector YY to XXXX
Action:  Reads a 256 byte sector from the logical drive to
         memory.


Command: WS   Write sector
Format:  Write sector to drive Z track YY sector YY from XXXX
Action:  Writes a sector from memory to the drive.


Command: TS   Test drive Stepping
Format:  Test stepping on drive Z    Hit any key to stop. . .
Action:  Selects and steps drive between track 00 and track 39 and
         back again. (unused for now as no FDC)


Command: DF   Disc Format to FLEX standard
Format:  Disc format on drive Z    Scratch disc in drive? T
Action:  Formats a disc to single sided, single density, 40 track FLEX
         standard (390 sectors free). Note that the date is not
         set, nor are the sectors tested. For now, this command is intended
         to format the F-RAMdisk


Command: TD   Test Drive
Format:  Random sector read on drive Z
Action:  Reads random sectors on the drive. As there is no FDC yet, this
         command is redundant as the silicon based drives always return
         with no error.
```

5.5 - FLEX related commands

```
Command: JF   Jump to FLEX warm start
Format:  Jump to flex warm start.....
Action:  Jumps to address $CD03
```

```
Command: BF   Boot FLEX from the system drive
Format:  Booting FLEX from system drive....
Action:  FLEX is loaded from logical drive 0 by firstly looking in the
         directory for either FLEX.SYS or FLEX.COR. Once loaded then the
         command overlays the console and disk jump tables, disables the
         date prompt, and sets the TTYSET and ASN parameters from the
         RTC/PRAM before jumping to the FLEX cold start point.
         Note that is not necessary to configure and link a version of FLEX
         as any copy of FLEX regardless of the machine it was designed to
         run on may be used.

Command: RM   Remote mount PROMdisk image
Format:  Remote mount PROMdisk image (MB2K2.DSK) to drive 0
Action:  Mount a FLEXNet connected remote disk image named MB2K2PD.DSK
         to logical drive 0. This is intended to act as a 'get out of jail
         free' card in case corruption of the PROMdisk image in flash stops
         the system from booting.
```

5.6 - OS-9 related commands

```
Command: BO   Boot OS-9 from the internal pre loaded image
Format:  Booting internal OS-9 L1.....
Action:  Set up the OS9 interrupt vectors, map in the shadow RAM and move
         the OS9 boot image with the OS9 kernel and 'ROM' memory resident
         modules from $0000 -> $C000, and start the kernel.
```

5.7 - Misc commands

The last two commands are concerned with testing memory and the real time clock.

```
Command: DC   Display RTC contents
Format: Display clock contents
Action: Displays the RTC ram in the following way:-

08:55:03  28/04/20                                  time and date
00 00 00 80                                         RTC control regs
0F                                                  FDC parameters
AA                                                  RTC powerful flag
00 01 FF FF                                         logical drive mapping
08 00 3A 18 50 00 00 08 00 00 1B                    FLEX 'TTYSET' params
00 01                                               FLEX 'ASN' parameters
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00        reserved for future use
00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF     user application bytes
```

```
Command: MC   Modify RTC
Format:  RTC examine and alter from YY
Action:  Examine and modify RTC ram contents in the same way as ME.
```

Section 6 - FLEX/OS-9 utilities

The section describes the utilities written to support specific features of the MB2K2. For FLEX the utilities are pre installed on the PROMdisk and also on MB2K2_SRC DSK in the MB2K2 utilities folder. OS-9 just uses versions of the FLEXNet utilities and these are found on the PROMdisk.

6.1 - FLEX specific utilities

TIME.CMD - This program gives the date/time from the RTC if installed. It also updates the FLEX date registers, and so should be included in your startup.txt file to give FLEX the correct date on boot.

SETTIME.CMD - This program sets the RTC time and date and is Y2K compliant.

ALLOCATE.CMD - In the MB2K2 there is no fixed relationship between the logical drive numbers of FLEX and the physical drives. The allocate command sets up this mapping. Running the command without any parameters will give the current mapping. Running the command followed by a space and then four characters from this list will set the allocation:-

0.....Drive type 0 is the PROMdisk
1.....Drive type 1 is the RAMdisk
2.....Drive type 2 is the F-RAMdisk
3.....Drive type 3 is the remote FLEXNet drive(s)

If a logical drive is not to be allocated then a '.' should be used. ie

‘ALLOCATE 0123’ would assign FLEX drive zero to the PROMdisk, drive 1 to the RAMdisk, the F-RAMdisk to drive two and a remote FLEXNet volume to drive three.

‘ALLOCATE 20.3’ would assign the F-RAMdisk to drive zero, the PROMdisk to drive 1. nothing to drive two, and and a remote FLEXNet volume to drive three.

Multiple FLEXNet drives may be allocated, so ‘ALLOCATE 3333’ would assign all four FLEX drives to FLEXNet. If the RTC/PRAM is not valid (due to failure of the battery supply) then the default allocation is (01..).

TTYSET.CMD and ASN.CMD - These are copies of the standard FLEX utilities which have been modified to save the current values to the RTC. In this way the TTYSET and ASN parameters will be set automatically when booting FLEX

FAST.CMD - Take control of the graphics terminal emulation from MON09 and passes it to the GDC’s internal emulator. This has the same effect as building MON09 using the fastest version of ‘GDCOUT.TXT’.

PDRW.CMD - Enables writing to the PROMdisk.

PDRO.CMD - Disables writing to the PROMdisk, this is the default.

FMTPD.CMD - Format PROMdisk allows increasing the PROMdisk size to 1.4MB ‘in situ’ Note that this command will erase the PROMdisk so should be used with care. If the PROMdisk becomes corrupted then the system can restored using the ‘RM’ command in MON09 to mount a remote version of the PROMdisk for recovery.

FLEXLINK.TXT & MONLINK.TXT - These files contain all of the FLEX and MON09 equates RMB's and jump table entries for use in your own programs. To use them in a program include the LIB pseudo op :-

```
 OPT NOL               SWITCH OFF LISTING
 LIB FLEXLINK          LINK IN FLEXLINK AND MONLINK
 OPT LIS               SWITCH ON LISTING
*
 ORG $C100             ORIGIN IS FLEX TPA
*
START JSR [CLEARG]     JUST CLEAR THE GRAPHICS SCREEN
 JMP FWARM             AND RETURN TO FLEX
*
 END START             TRANSFER ADDRESS IS START
```

Note that the monitor subroutine call is done by an indirect JSR. The source of flexlink and monlink is presented in appendix 2.

TYIO.TXT & STY-MB2.TXT - These two files are the drivers to adapt the word processing program STYLOGRAPH to the MB2K2. To adapt the program follow the procedure laid out in the STYLOGRAPH manual.

DEMO.CMD - This is a graphics demo for the MB2 which runs unmodified on the MB2K2.

TEXT.CMD - This program sets the display to the text screen.

GRAPH.CMD - This program sets the display to the graphics screen.

CLEARG.CMD - This command clears the graphics screen.

RECORD.CMD - This program will accept bytes from serial port 1 and store them into memory. The program assumes that the graphics display code of appendix 4 is being used.

INTERP.CMD - This program will decode the graphics display code in memory (starting from 0x0000), and display the commands and parameters on the text output device.

PLAY.CMD - This program will display the graphics images generated by the graphics display code in memory.

GRAPHICS.MAC - This is the graphics display code macro set. (see appendix 4 for more details)

The remaining utilities on the DSK are all modified versions of the utilities supplied as part of Michael Evenson's FLEXNet package for the Windows OS. Thee is no support for MacOS or Linux in this release but this is expected to be added in the future.

All of the utilities have been re-named to RXXX for consistency even though they support multiple mounted drives. Because the MB2K2 design has FLEXNet compatible drivers in ROM, the utilities do not need to check and link to drivers installed by the FNETDRV command, (which is no longer needed), instead the RMOUNT command has the network checks that were previously part of FNETDRV. The following descriptions are from the FLEXNet manual with additions for the MB2K2.

RCREATE - Remote Create

The RCREATE command will prompt the user for the location, name, and other parameters to create a new DSK file. After all parameters are entered, the user will be asked for a confirmation (yes/no) and if the answer is positive the file will be created. No parameters are required in the command line.

```
Example:
+++RCREATE
File path ? C:\folder3
File name ? newname
Disk number (in decimal) ? 123
Number of tracks (in decimal) ? 35
Sectors per track (in decimal) ? 10
OK to proceed ? Y

File successfully created
+++
```

NOTES:

- The file path is the directory where the new file will be created. If none is specified, the file will be created in the current directory.

- The file name ("newname") will be used for both the name of the DSK file, as known by Windows, and for the volume name which is stored in the SIR (track 0, sector 3) of the DSK file.

- The disk number, 123 in this example, is stored as the volume number in the SIR.

- If non-numeric values are entered for the disk number, the number of tracks or the number of sectors per track, the question will be repeated.

- RCREATE does not check for the "validity" of the track/sector format entered by the user. The file will be created, as long as the values given are numeric.

- In the above example, the disk image will be created with 35 tracks (including track 0) of 10 sectors, for a total of 340 data sectors.

- Note that RCREATing a disk does not automatically RMOUNT it or change the current directory. You must use RCD to change directories if needed, and RMOUNT to mount the new disk.

RDELETE - Remote Delete

RDELETE will prompt the user for the file path and the name of the file to be deleted. If no file path is given, the current directory will be assumed. RDELETE also adds a .DSK extension to the filename. RDELETE will then display the full path and name of the file, and will prompt the user for a response (Yes/No) before deleting the file. If the file cannot be found, an error message will be displayed.

Syntax: RDELETE <filename> [cr]

RLIST - Remote List

RLIST displays a list of all the directories which are contained in the current directory. It is provided as an aid to the user when "navigating" in the Windows directory tree structure. No parameters are needed. RLIST displays the listing in the same fashion as RDIR, i.e. 20 lines at a time, then prompting the user for more.

Syntax: RLIST [cr]

6.2 - OS-9 and FLEX utilities

RCD - Remote Change Directory

This command will change the current directory, which is the default directory which the other utilities will use. For example, if no file path is provided in the RCREATE command, the new file will be created in the current directory. If RCD is typed without parameters, the the current file path will be shown.

Syntax: RCD <filepath>

Example: RCD C:\folder3

RDIR - Remote Directory

RDIR performs a DIR command under DOS, and returns an exact copy of the listing generated on the MS-DOS screen:

- The current directory is used
- Only files with the extension .DSK will be listed.
- Command line parameters are allowed, and are passed to the Windows command.

- RDIR displays 20 lines at a time, then prompts the user before displaying the next 20 lines. Typing a space will display the next lines, typing [esc] will stop the listing and return to the FLEX prompt.

Examples
+++RDIR [cr] Lists all DSK files in the current directory

+++RDIR A2* [cr] Lists all DSK files which start with A2.

RESYNC - RE-SYNChronize

If something should happen and you restart the host computer, it will be looking for the Sync information from the FLEX/OS-9 system. RESYNC will re-establish the communication link and you can proceed. The command is simply RESYNC. If all is working properly you should not have to do anything at the PC end of the link.

Syntax: RESYNC [cr]

RMOUNT - Remote Mount

RMOUNT will "mount" a DSK file to the given drive number, in other words it selects a DSK file, opens it and sets it as the drive. After a file has been RMOUNTed, it can be read or written to by FLEX/OS-9 as it were a standard drive. RMOUNT assumes that the .DSK file is in the current Windows directory otherwise, the user may give the complete path and file name to RMOUNT a file which is in another directory. RMOUNT does not require a file extension to be typed; however, if one is entered, it must be .DSK (lowercase letters OK) or an error message will be generated. RMOUNT will check that the remote computer has FLEXNet running and will connect if needed.

NOTE: There is no need to "unmount" a file before RMOUNTing another one. However, a file which is "mounted" is open under Windows, and may be corrupted if the host computer is powered down while the file is open.

Examples:
+++RMOUNT <drive> <filename.dsk> [cr]
+++RMOUNT <drive> <filename> [cr]

Mount a file which is in the current directory. FLEXNet will automatically add the .DSK extension to the file name.

+++RMOUNT <drive> c:\folder\filename [cr]

Mount a file which is in any directory. In this case the file will be mounted without changing the current directory.

Note that for OS-9 <drive> is one of the /H0 through /H3 device paths. If <drive is not specified then /H0 is assumed.

For OS-9, RMOUNT is a memory resident module loaded at boot time and so may be used to connect to a remote drive even if there is no valid PROMdisk image.

REXIT - Remote Exit

This command will close all files open on the host computer and exit to Windows.

6.3 - OS-9 only utilities

blinky - simple multitasking test program

blinky will simply flash the green ‘6809’ LED approximately once per second. It’s only function is to test that interrupts and multitasking is running. Append a ‘&’ to the file name when launching from the Shell to make it a background task.

Syntax: blinky& [cr]

gotoxy - MB2K2 specific terminal emulator control code data file for the ‘Frank Hogg Dyna’ applications such as DynaStar.

H2 & H3 - are device descriptor for the last two FLEXLink drives (H0 and H0 are memory resident at boot time).

Section 7 - Programming guide

The subroutines in MON09 may be called from other programs by means of indirect JSRs via a table located at $F000. This table retains the same structure as MON09 for the MB2 for backwards compatibility and is included in the monlink.txt header file. The header files are listed in appendix 2 and can be found on the MB2K2_SRC DSK in this package.

To use these routines in your program, insert a LIB flexlink directive at the beginning, and the use an indirect jump to subroutine whenever a routine is used. ie

```
opt nol
lib flexlink
opt lis
  .
  .
ldx #100
ldy #354
jsr [LINE]
  .
  .
```

7.1 - MB2 MON09 general routines

STATUS
- Status routine.
- Entry: no parameters.
- Exit: (Z)=0 if character ready.

INCH1
- Input character with no echo and input.
- Entry: no parameters.
- Exit: (A) = character.

INCH
- Input character with echo INCH
- Entry: no parameters
- Exit: (A) = character.

OUTCH
- Output char.
- Entry: (A) = character.
- Exit: no change.

READ
- Read sector routine.
- Entry: (X) = address where sector is to be placed.
- (A) = Track number.
- (B) = Sector number.
- Exit: (B) = Error code (z)=1 if no error.

WRITE
- Write track routine.
- Entry: (X) = Address of area of memory from which the data will be taken.
- (A) = Track number.
- (B) = Sector number.
- Exit: (B) = Error condition, (Z)=1 no an error.

VERIFY
- Verify sector routine.
- Entry: no parameters.
- Exit: (B) = Error condition (Z)=1 if no error.

RST
- Restore drive to track 00.
- Entry: (X) = FCB address (3,X contains drive number).
- Exit: (B) = Error condition, (Z)=1 if no error.

DRV
- Select current drive.
- Entry: (X) = FCB address (3,X contains drive number).
- Exit: (B) = Error condition, (Z)=0 and (c)=1 if error.
- (B) = $0F if non existent drive.

CHKRDY
- Check for drive ready.
- Entry: (X) = FCB address (3,X contains drive number)>
- Exit: (B) = Error condition, (Z)=0 AND (C)=1 if drive is not ready.

QUICK
- Quick drive ready check.
- Entry: (X) = FCB address (3,X contains drive number).
- Exit: (B) = Error condition, (Z)=0 AND (c)=1 if drive not ready.

DINIT
- Init (cold start).
- Entry: no parameters.
- Exit: no change.

WARM
- Warm start.
- Entry: no parameters.
- Exit: no change.

SEEK
- Seek track.
- Entry: (A) = Track number.
- (B) = Sector number.
- Exit: (B) = Error condition, (Z)=1 if no error.

PCRLF
- Print a CR followed by a LF.
- Entry: no parameters.
- Exit: (A) destroyed.

PDATA1
- Print character string .
- Entry: (X) = Pointer to character string.
- Exit: (X) = Pointer to end of string token Hex(04).
- (A) Destroyed.

PSTRNG
- Print character string preceded by a CR,LF.
- Entry: (X) = Pointer to character string.
- Exit: (X) = Pointer to end of string token Hex(04).
- (A) = Destroyed.

PRINTA
- Print the A reg.
- Entry: (A) = Data to be printed.

PRINTX
- Print the X reg.
- Entry: (X) = Data to be printed.

DELAY
- Delay routine.
- Entry: (X) = Delay time in milli seconds.
- Exit: no change.

BADDR
- Build a four hex digit address.
- Entry: no parameters.
- Exit: (X) = Address.
- (A) = Destroyed.
- (B) = Destroyed.

BYTE
- Get a two digit hex byte.
- Entry: no parameters.
- Exit: (A) = Byte.

OUTHL
- Print left hex digit.
- Entry: (A) = Byte containing digit.
- Exit: (A) = Byte containing shifted digit.

OUTHR
- Output right hex digit.
- Entry: (A) = Byte containing digit.
- Exit: (A) = ASCII coded digit.

INHEX
- Input a valid hex character (If not hex then backspace).
- Entry: no parameters.
- Exit: (A) = Valid hex char.

OUT2H
OUT2HA
OUT4HS
OUT2HS
- Hex print routines.
- Entry: (X) = Pointer to a one or two byte hex number.
- Exit: (A) = Destroyed.

OUTS
- Output a space.
- Entry: no parameters.
- Exit (A) = Destroyed.

RANDOM
- Random number generator.
- Entry: no parameters.
- Exit: (A) = Random number from 0 to 255.

GETRTC
- Get a byte from the RTC.
- Entry : (B) = RTC address.
- Exit : (A) = Data.

PUTRTC
- Send a byte to the RTC.
- Entry : (B) = RTC address (A) = Data

BLEEP
- Beep for 100ms.

7.2 - MB2K2 MON09 additional routines

These routines are not in the original MB2 version.

TOUPPER
- convert to upper case chars in the range a-z
- Entry: (A) = ASCII char to be converted.
- Exit: (A) = converted char if in range, else no change.

BCD2BIN
- convert BCD coded value to binary
- Entry: (A) = value to be converted.
- Exit: (A) = converted value.

BIN2BCD
- convert binary value to BCD
- Entry: (A) = value to be converted.
- Exit: (A) = converted value.

7.3 - MB2/MB2K2 MON09 graphics routines

GCOM
- Send GDC command.
- Entry: (A) = GDC command
- Exit: No change.

GPRM
- Send GDC parameter.
- Entry: (A) = GDC parameter
- Exit: No change.

GPRMI
- Get a parameter from GDC.
- Entry: No parameters.
- Exit: (A) = Parameter byte

MASK
- Set mask.
- Entry: (X) = Mask value
- Exit: No change.

SETPEN
- Define line profile and 'pen' type.
- Entry: (A) = Pen type (0=replace 1=complement
  2=reset 3=set)
- (X) = Line profile
- Exit: No change.

SETPAT
- Set up graphics pattern in parameter ram.
- Entry: (X) = Pointer to eight byte pattern
- Exit: No change.

FIGSF
- Start figure drawing using parameter set in ram.
- Entry: (B) = Number of parameter bytes.

FIGSG
- Start graphics drawing using parameter set in ram.
- Entry: (B) = Number of parameter bytes.
- Exit: No change

SETPAR
- Set up display partitions in GDC.
- Entry: (X) = Start address of partition 1
- (D) = Start address of partition 2
- (Y) = Number of lines in partition 1
- (U) = Number of lines in partition 2
- Exit: No change.

SYNC
- Wait until vertical blanking period.
- Entry: No parameters.
- Exit: No change.

SETCRG
- Set graphics cursor.
- Entry: (X) = x coord (0<=x<=767)
- (Y) = y coord (0<=y<=575)
- Exit: No change.

GETCRG
- Read graphics cursor.
- Entry: No parameters.
- Exit: (X) = x coord of cursor
- (Y) = y coord of cursor

OFF
- Switch off display.
- Entry: No parameters.
- Exit: No change.

ON
- Switch on display.
- Entry: No parameters.
- Exit: No change.

GRAPH
- Set display to graphics.
- Entry: No parameters.
- Exit: No change.

MODE
- Set GDC mode.
- Entry: (A) = New mode byte
- (B) = Read flag
- Exit: If (B) <> 0 then (A) = New mode byte
- If (B) = 0 then (A) = OLD mode byte

ZOOM
- Set graphics zoom.
- Entry: (A) = New zoom byte
- (B) = Read flag
- Exit: If (B) <> 0 then (A) = New zoom byte
- If (B) = 0 then (A) = OLD zoom byte

FILL
- Area fill.
- Entry: (A) = Initial drawing direction
- (X) = Number of pixels in the initial direction
- (Y) = Number of pixels in the perpendicular direction
- Exit: No change.

CLEARG
- Clear graphics screen.
- Entry: No parameters.
- Exit: No change.

CLEARX
- Clear GDC ram from current cursor.
- Entry: (A) = Drawing mode (0=replace 1=complement 2=reset 3=set)
- (X) = Number of words to be cleared
- Exit: No change

GDCINIT
- Init display.
- Entry: No parameters.
- Exit: No change.

POINT
- Plot a point at the current cursor position.
- Entry: No parameters.
- Exit: No change.

LINE
- Plot a line from the current cursor portion.
- Entry: (X) = x coord
- (Y) = • Entry: coord
- Exit: No change.

RECT
- Plot a rectangle.
- Entry: (A) = Initial drawing direction
- (X) = Length of side in the initial direction
- (Y) = Length of side in the perpendicular direction
- Exit: No change.

CIRCLE
- Plot a circle at the current cursor location.
- Entry: (A) = radius of circle (0<A<127)
- Exit: No change.

SETCRT
- Set text cursor.
- Entry: (X) = Cursor word address
- Exit: No change.

GETCRT
- Get text cursor.
- Entry: No parameters.
- Exit: (X) = Cursor word address

TEXT
- Set display to text.
- Entry: No parameters.
- Exit: No change.

CLEART
- Clear text screen.
- Entry: No parameters.
- Exit: No change.

GDCOUT
- Put ascii character to screen.
- Entry: (A) = Character
  (control codes are given in appendix 1)

Appendix 1 - Terminal emulator control codes

DECIMAL	HEX	KEY	FUNCTION
0	00	^@	NULL
1	01	^A	-
2	02	^B	-
3	03	^C	-
4	04	^D	EOT
5	05	^E	-
6	06	^F	-
7	07	^G	BELL
8	08	^H	BACKSPACE (CURSOR LEFT)
9	09	^I	CURSOR RIGHT
10	0A	^J	LINE FEED (CURSOR DOWN)
11	0B	^K	CURSOR UP
12	0C	^L	CLEAR SCREEN
13	0D	^M	RETURN
14	0E	^N	MOVE CURSOR (SEE NOTE)
15	0F	^O	HOME
16	10	^P	SCREEN ON
17	11	^Q	SCREEN OFF
18	12	^R	CURSOR ON
19	13	^S	CURSOR OFF
20	14	^T	SET CURSOR TYPE 1
21	15	^U	SET CURSOR TYPE 2
22	16	^V	INVERT ON
23	17	^W	INVERT OFF
24	18	^X	ERASE TO END OF LINE
25	19	^Y	ERASE TO END OF SCREEN
26	1A	^Z	ERASE LINE
27	1B	-	ESCAPE
28	1C	-	-
29	1D	-	-
30	1E	-	-
31	1F	-	-

NOTE: Move cursor has two parameters. The control code should be followed by two bytes, row and column. The home position is 0,0. The value 0x20 should be added to each value. ie to move the cursor to row 4 col 7, send the byte sequence 0x0E,0x24,0x27.

Appendix 2 - Flexlink and Monlink sources

2.1 - Monlink

```
      * Monitor definitions and equates for the MB2K2.
E000  PROM    EQU    $E000     Eprom start address.
DE00  RAM     EQU    $DE00     Scratch ram + stack space.
FF00  IO      EQU    $FF00     I/O base address.
DE6F  SSTACK  EQU    (RAM+127-16) Top of system stack.
DF80  SCRAT   EQU    (RAM+384) Start of scratch space.
      *
      * User callable subroutines. Use indirect JSR's to call.
F000  RESET   EQU    $F000     Cold start.
F002  CONTRL  EQU    $F002     Warm start.
F004  INCH1   EQU    $F004     Input char without an echo .
F006  INCH    EQU    $F006     Input char .
F008  STATUS  EQU    $F008     Check for char.
F00A  OUTCH   EQU    $F00A     Output char.
F00C  PDATA1  EQU    $F00C     Print string terminated by hex(04).
F00E  PCRLF   EQU    $F00E     Print a cr followed by a lf.
F010  PSTRNG  EQU    $F010     PCRLF followed by PDATA1.
F012  INIT    EQU    $F012     Init active device.
F014  DELAY   EQU    $F014     Delay for (XREG) m/S.
F016  BADDR   EQU    $F016     Get a four digit hex address into X.
F018  BYTE    EQU    $F018     Get a two hex digit number into A.
F01A  INHEX   EQU    $F01A     Get a one digit hex char into A.
F01C  OUT2H   EQU    $F01C     Output two hex chars pointed to by X.
F01E  OUT2HS  EQU    $F01E     OUT2H plus a space.
F020  OUT4HS  EQU    $F020     Output four hex chars etc.
F022  OUTHR   EQU    $F022     Output right hex digit in A.
F024  OUTHL   EQU    $F024     Output left hex digit in A.
F026  OUTS    EQU    $F026     Output a space.
F028  RANDOM  EQU    $F028     Returns a random number in the range 0-255.
F02A  PRINTA  EQU    $F02A     Print the contents of A.
F02C  PRINTX  EQU    $F02C     Print the contents of X.
F02E  READ    EQU    $F02E     Read sector.
F030  WRITE   EQU    $F030     Write sector.
F032  VERIFY  EQU    $F032     Verify sector.
F034  RST     EQU    $F034     Restore to track 00.
F036  DRV     EQU    $F036     Select drive.
F038  CHKRDY  EQU    $F038     Check for drive ready.
F03A  QUICK   EQU    $F03A     Quick check for drive ready.
F03C  DINIT   EQU    $F03C     Drive cold start.
F03E  WARM    EQU    $F03E     Drive warm start.
F040  SEEK    EQU    $F040     Seek to track.
F042  GETTIM  EQU    $F042     Get time string from RTC.
F044  PUTTIM  EQU    $F044     Put time string to RTC.
F046  GETRTC  EQU    $F046     Get a byte from the RTC.
F048  PUTRTC  EQU    $F048     Put a byte to the RTC.
F04A  BEEP    EQU    $F04A     Sound a 100ms tone.
F04C  GCOM    EQU    $F04C     Send command to GDC.
F04E  GPRM    EQU    $F04E     Send parameter to GDC.
F050  GPRMI   EQU    $F050     Get parameter from GDC.
F052  MASK    EQU    $F052     Load mask register.
F054  SETPEN  EQU    $F054     Define drawing mode.
F056  SETPAT  EQU    $F056     Define graphics pattern.
F058  FIGSF   EQU    $F058     Start figure drawing.
F05A  FIGSG   EQU    $F05A     Start graphics drawing.
F05C  SETPAR  EQU    $F05C     Define display partitions.
F05E  SETCRG  EQU    $F05E     Set graphics cursor.
F060  GETCRG  EQU    $F060     Get graphics cursor.
F062  SETCRT  EQU    $F062     Set text cursor.
F064  GETCRT  EQU    $F064     Get text cursor.
F066  OFF     EQU    $F066     Turn display off.
F068  ON      EQU    $F068     Turn display on.
F06A  GRAPH   EQU    $F06A     Set display to graphics.
F06C  TEXT    EQU    $F06C     Set display to text
F06E  MODE    EQU    $F06E     Set GDC mode.
```

```
F070  ZOOM    EQU    $F070     Set zoom factors.
F072  FILL    EQU    $F072     Area fill routine.
F074  CLEARX  EQU    $F074     Clear X words of display memory.
F076  CLEARG  EQU    $F076     Clear graphics display.
F078  CLEART  EQU    $F078     Clear text display.
F07A  GDCINIT EQU    $F07A     Initialise GDC.
F07C  GDCOUT  EQU    $F07C     Output a character.
F07E  INKEY   EQU    $F07E     Get a character from the keyboard.
F080  POINT   EQU    $F080     Plot a point.
F082  LINE    EQU    $F082     Plot a line.
F084  RECT    EQU    $F084     Plot a rectangle.
F086  CIRCLE  EQU    $F086     Plot a circle.
F088  ARC     EQU    $F088     Plot an arc
F08A  CLINK   EQU    $F08A     Link text parameters.
F08C  SYNC    EQU    $F08C     Sync to vertical blanking.
      *
      * the following routines were not in the MB2 ROM.
F08E  TOUPP   EQU    $F08E     convert ASCII char to upper case
F090  BCD2B   EQU    $F090     convert BCD value to binary
F092  BIN2B   EQU    $F092     convert binary value to BCD
      *
F0A0  SCHAR   EQU    $F0A0     FLEXNet driver send char
F0A2  RCHAR   EQU    $F0A2     FLEXNet driver receive char
      *
      *
DE80                 ORG    (RAM+128)
DE80           BUFFER RMB    256       Floppy interface sector buffer.
DF80           STACK  RMB    2         User system stack.
DF82           NMIV   RMB    2         NMI interrupt vector.
DF84           IRQV   RMB    2         IRQ interrupt vector.
DF86           FIRQV  RMB    2         FIRQ interrupt vector.
DF88           SWI2V  RMB    2         SWI2 interrupt vector.
DF8A           SWI3V  RMB    2         SWI3 interrupt vector.
DF8C           IPORT  RMB    1         Active input port.
DF8D           OPORT  RMB    1         Active output port.
DF8E           DRIVE  RMB    1         Format drive value.
DF8F           TRACK  RMB    1         Format track value.
DF90           SECTOR RMB    1         Format sector value.
DF91           TEMP   RMB    1
DF92           XTEMP  RMB    2
DF94           YTEMP  RMB    2
DF96           TTO    RMB    2
DF98           RNDM   RMB    4         Random number storage.
DF9C           WARMS  RMB    1         Warm start flag.
DF9D           DDSTAB RMB    4         Disc driver type table.
DFA1           REAVEC RMB    2         Disc driver jump table.
DFA3           WRIVEC RMB    2
DFA5           VERVEC RMB    2
DFA7           RSTVEC RMB    2
DFA9           DRVVEC RMB    2
DFAB           CHKVEC RMB    2
DFAD           QUIVEC RMB    2
DFAF           INIVEC RMB    2
DFB1           WARVEC RMB    2
DFB3           SEEVEC RMB    2
DFB5           RTCFAIL RMB   1         RTC fail flag.
DFB6           CURDRV RMB    1         Active floppy drive.
DFB7           XCOORD RMB    2         Cursor X value.
DFB9           YCOORD RMB    2         Cursor Y Value.
DFBB           PART1  RMB    4         Display partition one.
DFBF           PART2  RMB    4         Display partition two.
DFC3           GPARAM RMB    8         Parameter ram contents.
DFCB           GMODE  RMB    1         GDC mode register contents.
DFCC           GZOOM  RMB    1         Display + write zoom values.
DFCD           GFIGS  RMB    1         Figs 1st parameter value.
DFCE           DC     RMB    2
DFD0           D      RMB    2
DFD2           D2     RMB    2
DFD4           D1     RMB    2
```

```
DFD6              DM      RMB    2
DFD8              CONST   RMB    1
DFD9              ROW     RMB    1
DFDA              COL     RMB    1
DFDB              MAXCOL  RMB    1
DFDC              MAXROW  RMB    1
DFDD              CCOL    RMB    1
DFDE              CROW    RMB    1
DFDF              BCOL    RMB    1
DFE0              BROW    RMB    1
DFE1              ATTRI   RMB    1
DFE2              CSPACE  RMB    1
DFE3              CHARTAB RMB    2
DFE5              CURSOR  RMB    2         Text cursor position.
DFE7              OFFSET  RMB    2
DFE9              CZOOM   RMB    1
DFEA              CTYPE   RMB    1
DFEB              ESCFLG  RMB    1
DFEC              TS1     RMB    2
DFEE              TS2     RMB    2
DFF0              TL1     RMB    2
DFF2              TL2     RMB    2
DFF4              DEN     RMB    1
DFF5              DEN1    RMB    1
DFF6              STEP    RMB    1
DFF7              STEP0   RMB    1
DFF8              SPEED   RMB    1
DFF9              FLASH   RMB    1
DFFA              CFLAG   RMB    1
                  *
                  * the following were not in the MB2 ROM.
DFFB              CHKSUM  RMB    2         FLEXNet driver checksum
DFFD              BRKPNT  RMB    1         saved instruction byte from breakpoint
DFFE              PDWPRT  RMB    1         promdisk write protect flag (0 = protected)
                  *
                  * Hardware device equates.
            FF00  KEYREG  EQU    $FF00     Keyboard register.
            FF01  PIACA   EQU    $FF01     Pia side a control register.
            FF02  SYSREG  EQU    $FF02     System control register.
            FF03  PIACB   EQU    $FF03     Pia side b control register.
                  *
            FF08  ACIAD1  EQU    $FF08     Acia port 0 data register.
            FF09  ACIAC1  EQU    $FF09     Acia port 0 control register.
            FF04  ACIAD2  EQU    $FF04     Acia port 1 data register.
            FF05  ACIAC2  EQU    $FF05     Acia port 1 control register.
            FF0C  BAUD1   EQU    $FF0C     Acia port 0 baud rate register.
            FF0D  BAUD2   EQU    $FF0D     Acia port 1 baud rate register.
                  *
            FF10  COMREG  EQU    $FF10     Fdc command register.
            FF11  TRKREG  EQU    $FF11     Fdc track register.
            FF12  SECREG  EQU    $FF12     Fdc sector register.
            FF13  DATREG  EQU    $FF13     Fdc data register.
                  *
            FF14  GDCPRM  EQU    $FF14     Gdc command register.
            FF15  GDCCOM  EQU    $FF15     Gdc parameter register.
                  *
            FF18  RTCADD  EQU    $FF18     Rtc address register.
            FF19  RTCDAT  EQU    $FF19     Rtc data register.
                  *
                  * ramdisk controller registers
            FF20  RCOMRG  EQU    $FF20
            FF21  RTRKRG  EQU    $FF21
            FF22  RSECRG  EQU    $FF22
            FF23  RDATRG  EQU    $FF23
                  *
                  * promdisk controller registers
            FF30  ECOMRG  EQU    $FF30
            FF31  ETRKRG  EQU    $FF31
            FF32  ESECRG  EQU    $FF32
```

```
FF33  EDATRG  EQU    $FF33
      *
      * disk controller commands
0000  RSCMD   EQU    $00
0010  SECMD   EQU    $10
0084  RECMD   EQU    $84
00A4  WRCMD   EQU    $A4
      *
      * Floppy disk controller status bits
0058  RSMASK  EQU    $58
0010  SEMASK  EQU    $10
001C  REMASK  EQU    $1C
005C  WRMASK  EQU    $5C
0018  VEMASK  EQU    $18
0002  DRQ     EQU    $2
0001  BUSY    EQU    $1
```

2.2 - FLEXlink

```
      *
      ******************************************
      * This file contains the subroutine and     *
      * storage location equates for FLEX. To     *
      * use this file insert the following lines  *
      * of code in your program :-                *
      *          OPT NOL                          *
      *          LIB FLEXLINK                     *
      *          OPT LIS                          *
      * For details of the routines and           *
      * parameters see the FLEX programmers guide *
      ******************************************
      *
      * Storage locations.
C080  LINBUF  EQU    $C080     Line buffer start.
CC00  TTYBS   EQU    $CC00     TTYSET backspace character.
CC0B  SYSDRV  EQU    $CC0B     System drive number.
CC0C  WRKDRV  EQU    $CC0C     Working drive number.
CC0E  MONTH   EQU    $CC0E     FLEX system date.
CC0F  DAY     EQU    $CC0F
CC10  YEAR    EQU    $CC10
CC2B  MEMEND  EQU    $CC2B     Memory end pointer.
      *
      * User callable routines.
CD00  FCOLD   EQU    $CD00     Cold start.
CD03  FWARM   EQU    $CD03     Warm start.
CD06  RENTER  EQU    $CD06     Main loop entry point.
CD4B  DOCMND  EQU    $CD4B     Call dos as a subroutine.
CD4E  STAT    EQU    $CD4E     Check terminal status.
CD09  FINCH   EQU    $CD09     Input character.
CD0C  INCH2   EQU    $CD0C     Input character switched.
CD0F  FOUTCH  EQU    $CD0F     Output character.
CD12  OUTCH2  EQU    $CD12     Output character switched.
CD15  GETCHR  EQU    $CD15     Get a char (main routine).
CD18  PUTCHR  EQU    $CD18     Put a char (main routine).
CD1B  INBUFF  EQU    $CD1B     Input into line buffer.
CD1E  FPSTRNG EQU    $CD1E     Print a char string.
CD21  CLASS   EQU    $CD21     Classify a char.
CD24  FPCRLF  EQU    $CD24     Print a crlf.
CD27  NXTCH   EQU    $CD27     Get next buffer char.
CD2A  RSTIO   EQU    $CD2A     Restore i/o vectors.
CD2D  GETFIL  EQU    $CD2D     Get file spec.
CD30  LOAD    EQU    $CD30     File loader.
CD33  SETEXT  EQU    $CD33     Set file extension.
CD39  OUTDEC  EQU    $CD39     Output decimal number.
CD3C  OUTHEX  EQU    $CD3C     Output hexadecimal number.
CD45  OUTADR  EQU    $CD45     Output hex address.
CD3F  RPTERR  EQU    $CD3F     Report error.
```

```
CD42  GETHEX  EQU    $CD42     Get hexadecimal number.
CD48  INDEC   EQU    $CD48     Input decimal number.
      *
      * Monitor definitions and equates for the MB2K2.
E000  PROM    EQU    $E000     Eprom start address.
DE00  RAM     EQU    $DE00     Scratch ram + stack space.
FF00  IO      EQU    $FF00     I/O base address.
DE6F  SSTACK  EQU    (RAM+127-16) Top of system stack.
DF80  SCRAT   EQU    (RAM+384) Start of scratch space.
      *
      * User callable subroutines. Use indirect JSR's to call.
F000  RESET   EQU    $F000     Cold start.
F002  CONTRL  EQU    $F002     Warm start.
F004  INCH1   EQU    $F004     Input char without an echo .
F006  INCH    EQU    $F006     Input char .
F008  STATUS  EQU    $F008     Check for char.
F00A  OUTCH   EQU    $F00A     Output char.
F00C  PDATA1  EQU    $F00C     Print string terminated by hex(04).
F00E  PCRLF   EQU    $F00E     Print a cr followed by a lf.
F010  PSTRNG  EQU    $F010     PCRLF followed by PDATA1.
F012  INIT    EQU    $F012     Init active device.
F014  DELAY   EQU    $F014     Delay for (XREG) m/S.
F016  BADDR   EQU    $F016     Get a four digit hex address into X.
F018  BYTE    EQU    $F018     Get a two hex digit number into A.
F01A  INHEX   EQU    $F01A     Get a one digit hex char into A.
F01C  OUT2H   EQU    $F01C     Output two hex chars pointed to by X.
F01E  OUT2HS  EQU    $F01E     OUT2H plus a space.
F020  OUT4HS  EQU    $F020     Output four hex chars etc.
F022  OUTHR   EQU    $F022     Output right hex digit in A.
F024  OUTHL   EQU    $F024     Output left hex digit in A.
F026  OUTS    EQU    $F026     Output a space.
F028  RANDOM  EQU    $F028     Returns a random number in the range 0-255.
F02A  PRINTA  EQU    $F02A     Print the contents of A.
F02C  PRINTX  EQU    $F02C     Print the contents of X.
F02E  READ    EQU    $F02E     Read sector.
F030  WRITE   EQU    $F030     Write sector.
F032  VERIFY  EQU    $F032     Verify sector.
F034  RST     EQU    $F034     Restore to track 00.
F036  DRV     EQU    $F036     Select drive.
F038  CHKRDY  EQU    $F038     Check for drive ready.
F03A  QUICK   EQU    $F03A     Quick check for drive ready.
F03C  DINIT   EQU    $F03C     Drive cold start.
F03E  WARM    EQU    $F03E     Drive warm start.
F040  SEEK    EQU    $F040     Seek to track.
F042  GETTIM  EQU    $F042     Get time string from RTC.
F044  PUTTIM  EQU    $F044     Put time string to RTC.
F046  GETRTC  EQU    $F046     Get a byte from the RTC.
F048  PUTRTC  EQU    $F048     Put a byte to the RTC.
F04A  BEEP    EQU    $F04A     Sound a 100ms tone.
F04C  GCOM    EQU    $F04C     Send command to GDC.
F04E  GPRM    EQU    $F04E     Send parameter to GDC.
F050  GPRMI   EQU    $F050     Get parameter from GDC.
F052  MASK    EQU    $F052     Load mask register.
F054  SETPEN  EQU    $F054     Define drawing mode.
F056  SETPAT  EQU    $F056     Define graphics pattern.
F058  FIGSF   EQU    $F058     Start figure drawing.
F05A  FIGSG   EQU    $F05A     Start graphics drawing.
F05C  SETPAR  EQU    $F05C     Define display partitions.
F05E  SETCRG  EQU    $F05E     Set graphics cursor.
F060  GETCRG  EQU    $F060     Get graphics cursor.
F062  SETCRT  EQU    $F062     Set text cursor.
F064  GETCRT  EQU    $F064     Get text cursor.
F066  OFF     EQU    $F066     Turn display off.
F068  ON      EQU    $F068     Turn display on.
F06A  GRAPH   EQU    $F06A     Set display to graphics.
F06C  TEXT    EQU    $F06C     Set display to text
F06E  MODE    EQU    $F06E     Set GDC mode.
F070  ZOOM    EQU    $F070     Set zoom factors.
F072  FILL    EQU    $F072     Area fill routine.
```

```
F074  CLEARX  EQU    $F074     Clear X words of display memory.
F076  CLEARG  EQU    $F076     Clear graphics display.
F078  CLEART  EQU    $F078     Clear text display.
F07A  GDCINIT EQU    $F07A     Initialise GDC.
F07C  GDCOUT  EQU    $F07C     Output a character.
F07E  INKEY   EQU    $F07E     Get a character from the keyboard.
F080  POINT   EQU    $F080     Plot a point.
F082  LINE    EQU    $F082     Plot a line.
F084  RECT    EQU    $F084     Plot a rectangle.
F086  CIRCLE  EQU    $F086     Plot a circle.
F088  ARC     EQU    $F088     Plot an arc
F08A  CLINK   EQU    $F08A     Link text parameters.
F08C  SYNC    EQU    $F08C     Sync to vertical blanking.
      *
      * the following routines were not in the MB2 ROM.
F08E  TOUPP   EQU    $F08E     convert ASCII char to upper case
F090  BCD2B   EQU    $F090     convert BCD value to binary
F092  BIN2B   EQU    $F092     convert binary value to BCD
      *
F0A0  SCHAR   EQU    $F0A0     FLEXNet driver send char
F0A2  RCHAR   EQU    $F0A2     FLEXNet driver receive char
      *
```

Appendix 3 - PS/2 keyboard mapping

```
// Keyboard mapping for PS-2 keyboard and MB2K2.
// First dataset, shift = 0, shiftlock = 0
 0x00    //Scancode = $00
 ,0x05   //Scancode = $01          F9      CEDRIC - delete char
 ,0x00   //Scancode = $02
 ,0x19   //Scancode = $03          F5      CEDRIC - cut
 ,0x0E   //Scancode = $04          F3      CEDRIC - search and replace
 ,0x03   //Scancode = $05          F1      CEDRIC - search forward
 ,0x1A   //Scancode = $06          F2      CEDRIC - search back
 ,0x08   //Scancode = $07          F12     CEDRIC - delete left (backspace)
 ,0x00   //Scancode = $08
 ,0x17   //Scancode = $09          F10     CEDRIC - delete word
 ,0x13   //Scancode = $0A          F8      CEDRIC -
 ,0x10   //Scancode = $0B          F6      CEDRIC - paste
 ,0x01   //Scancode = $0C          F4      CEDRIC - global replace
 ,0x09   //Scancode = $0D          Tab
 ,'`'    //Scancode = $0E          `
 ,0x00   //Scancode = $0F

 ,0x00   //Scancode = $10
 ,0x00   //Scancode = $11
 ,0x00   //Scancode = $12
 ,0x00   //Scancode = $13
 ,0x00   //Scancode = $14
 ,'q'    //Scancode = $15          q
 ,'1'    //Scancode = $16          1
 ,0x00   //Scancode = $17
 ,0x00   //Scancode = $18
 ,0x00   //Scancode = $19
 ,'z'    //Scancode = $1A          z
 ,'s'    //Scancode = $1B          s
 ,'a'    //Scancode = $1C          a
 ,'w'    //Scancode = $1D          w
 ,'2'    //Scancode = $1E          2
 ,0x00   //Scancode = $1F

 ,0x00   //Scancode = $20
 ,'c'    //Scancode = $21          c
 ,'x'    //Scancode = $22          x
 ,'d'    //Scancode = $23          d
 ,'e'    //Scancode = $24          e
 ,'4'    //Scancode = $25          4
 ,'3'    //Scancode = $26          3
 ,0x00   //Scancode = $27
 ,0x00   //Scancode = $28
 ,' '    //Scancode = $29          Space
 ,'v'    //Scancode = $2A          v
 ,'f'    //Scancode = $2B          f
 ,'t'    //Scancode = $2C          t
 ,'r'    //Scancode = $2D          r
 ,'5'    //Scancode = $2E          5
 ,0x00   //Scancode = $2F

 ,0x00   //Scancode = $30
 ,'n'    //Scancode = $31          n
 ,'b'    //Scancode = $32          b
 ,'h'    //Scancode = $33          h
 ,'g'    //Scancode = $34          g
 ,'y'    //Scancode = $35          y
 ,'6'    //Scancode = $36          6
 ,0x00   //Scancode = $37
 ,0x00   //Scancode = $38
 ,0x00   //Scancode = $39
 ,'m'    //Scancode = $3A          m
 ,'j'    //Scancode = $3B          j
 ,'u'    //Scancode = $3C          u
 ,'7'    //Scancode = $3D          7
```

```
,'8'    //Scancode = $3E          8
,0x00   //Scancode = $3F

,0x00   //Scancode = $40
,','    //Scancode = $41          ,
,'k'    //Scancode = $42          k
,'i'    //Scancode = $43          i
,'o'    //Scancode = $44          o
,'0'    //Scancode = $45          0
,'9'    //Scancode = $46          9
,0x00   //Scancode = $47
,0x00   //Scancode = $48
,'.'    //Scancode = $49          .
,'/'    //Scancode = $4A          /
,'l'    //Scancode = $4B          l
,0x3B   //Scancode = $4C          ;
,'p'    //Scancode = $4D          p
,'-'    //Scancode = $4E          -
,0x00   //Scancode = $4F

,0x00   //Scancode = $50
,0x00   //Scancode = $51
,0x27   //Scancode = $52          '
,0x00   //Scancode = $53
,'['    //Scancode = $54          [
,'='    //Scancode = $55          =
,0x00   //Scancode = $56
,0x00   //Scancode = $57
,0x00   //Scancode = $58
,0x00   //Scancode = $59
,0x0D   //Scancode = $5A          Enter
,']'    //Scancode = $5B          ]
,0x00   //Scancode = $5C
,0x5C   //Scancode = $5D          backslash
,0x00   //Scancode = $5E
,0x00   //Scancode = $5F

,0x00   //Scancode = $60
,0x00   //Scancode = $61
,0x00   //Scancode = $62
,0x00   //Scancode = $63
,0x00   //Scancode = $64
,0x00   //Scancode = $65
,0x08   //Scancode = $66          Backspace
,0x00   //Scancode = $67
,0x00   //Scancode = $68
,0x0A   //Scancode = $69          KP1     CEDRIC - move to line end
,0x00   //Scancode = $6A
,0x0C   //Scancode = $6B          KP4     CEDRIC - move cursor left
,0x09   //Scancode = $6C          KP7     CEDRIC - (Tab)
,0x00   //Scancode = $6D
,0x00   //Scancode = $6E
,0x00   //Scancode = $6F

,0x00   //Scancode = $70          KP0
,'.'    //Scancode = $71          KP.
,0x04   //Scancode = $72          KP2     CEDRIC - move cursor down
,0x1C   //Scancode = $73          KP5     CEDRIC - point here
,0x12   //Scancode = $74          KP6     CEDRIC - move cursor right
,0x15   //Scancode = $75          KP8     CEDRIC - move cursor up
,0x1B   //Scancode = $76          Escape
,0x00   //Scancode = $77
,0x18   //Scancode = $78          F11     CEDRIC - delete line
,'+'    //Scancode = $79          KP+
,0x06   //Scancode = $7A          KP3     CEDRIC - move one line forward
,'-'    //Scancode = $7B          KP-
,'*'    //Scancode = $7C          KP*
,0x02   //Scancode = $7D          KP9     CEDRIC - line back
,0x00   //Scancode = $7E
```

```
 ,0x00  //Scancode = $7F

// Second dataset, shift = 0, shiftlock = 1
 ,0x00  //Scancode = $00
 ,0x05  //Scancode = $01          F9      CEDRIC - delete char
 ,0x00  //Scancode = $02
 ,0x19  //Scancode = $03          F5      CEDRIC - cut
 ,0x0E  //Scancode = $04          F3      CEDRIC - search and replace
 ,0x03  //Scancode = $05          F1      CEDRIC - search forward
 ,0x1A  //Scancode = $06          F2      CEDRIC - search back
 ,0x08  //Scancode = $07          F12     CEDRIC - delete left (backspace)
 ,0x00  //Scancode = $08
 ,0x17  //Scancode = $09          F10     CEDRIC - delete word
 ,0x13  //Scancode = $0A          F8      CEDRIC -
 ,0x10  //Scancode = $0B          F6      CEDRIC - paste
 ,0x01  //Scancode = $0C          F4      CEDRIC - global replace
 ,0x09  //Scancode = $0D          Tab
 ,'~'   //Scancode = $0E          `
 ,0x00  //Scancode = $0F

 ,0x00  //Scancode = $10
 ,0x00  //Scancode = $11
 ,0x00  //Scancode = $12
 ,0x00  //Scancode = $13
 ,0x00  //Scancode = $14
 ,'Q'   //Scancode = $15          Q
 ,'1'   //Scancode = $16          1
 ,0x00  //Scancode = $17
 ,0x00  //Scancode = $18
 ,0x00  //Scancode = $19
 ,'Z'   //Scancode = $1A          Z
 ,'S'   //Scancode = $1B          S
 ,'A'   //Scancode = $1C          A
 ,'W'   //Scancode = $1D          W
 ,'2'   //Scancode = $1E          2
 ,0x00  //Scancode = $1F

 ,0x00  //Scancode = $20
 ,'C'   //Scancode = $21          C
 ,'X'   //Scancode = $22          X
 ,'D'   //Scancode = $23          D
 ,'E'   //Scancode = $24          E
 ,'4'   //Scancode = $25          4
 ,'3'   //Scancode = $26          3
 ,0x00  //Scancode = $27
 ,0x00  //Scancode = $28
 ,' '   //Scancode = $29          Space
 ,'V'   //Scancode = $2A          V
 ,'F'   //Scancode = $2B          F
 ,'T'   //Scancode = $2C          T
 ,'R'   //Scancode = $2D          R
 ,'5'   //Scancode = $2E          5
 ,0x00  //Scancode = $2F

 ,0x00  //Scancode = $30
 ,'N'   //Scancode = $31          N
 ,'B'   //Scancode = $32          B
 ,'H'   //Scancode = $33          H
 ,'G'   //Scancode = $34          G
 ,'Y'   //Scancode = $35          Y
 ,'6'   //Scancode = $36          6
 ,0x00  //Scancode = $37
 ,0x00  //Scancode = $38
 ,0x00  //Scancode = $39
 ,'M'   //Scancode = $3A          M
 ,'J'   //Scancode = $3B          J
 ,'U'   //Scancode = $3C          U
 ,'7'   //Scancode = $3D          7
 ,'8'   //Scancode = $3E          8
```

```
,0x00  //Scancode = $3F

,0x00  //Scancode = $40
,','   //Scancode = $41          ,
,'K'   //Scancode = $42          K
,'I'   //Scancode = $43          I
,'O'   //Scancode = $44          O
,'0'   //Scancode = $45          0
,'9'   //Scancode = $46          9
,0x00  //Scancode = $47
,0x00  //Scancode = $48
,'.'   //Scancode = $49          .
,'/'   //Scancode = $4A          /
,'L'   //Scancode = $4B          L
,0x3B  //Scancode = $4C          ;
,'P'   //Scancode = $4D          P
,'-'   //Scancode = $4E          -
,0x00  //Scancode = $4F

,0x00  //Scancode = $50
,0x00  //Scancode = $51
,0x27  //Scancode = $52          '
,0x00  //Scancode = $53
,'['   //Scancode = $54          [
,'='   //Scancode = $55          =
,0x00  //Scancode = $56
,0x00  //Scancode = $57
,0x00  //Scancode = $58
,0x00  //Scancode = $59
,0x0D  //Scancode = $5A          Enter
,']'   //Scancode = $5B          ]
,0x00  //Scancode = $5C
,0x5C  //Scancode = $5D          backslash
,0x00  //Scancode = $5E
,0x00  //Scancode = $5F

,0x00  //Scancode = $60
,0x00  //Scancode = $61
,0x00  //Scancode = $62
,0x00  //Scancode = $63
,0x00  //Scancode = $64
,0x00  //Scancode = $65
,0x08  //Scancode = $66          Backspace
,0x00  //Scancode = $67
,0x00  //Scancode = $68
,0x0A  //Scancode = $69          KP1     CEDRIC - move to line end
,0x00  //Scancode = $6A
,0x0C  //Scancode = $6B          KP4     CEDRIC - move cursor left
,0x09  //Scancode = $6C          KP7     CEDRIC - (Tab)
,0x00  //Scancode = $6D
,0x00  //Scancode = $6E
,0x00  //Scancode = $6F

,0x00  //Scancode = $70          KP0
,'.'   //Scancode = $71          KP.
,0x04  //Scancode = $72          KP2     CEDRIC - move cursor down
,0x1C  //Scancode = $73          KP5     CEDRIC - point here
,0x12  //Scancode = $74          KP6     CEDRIC - move cursor right
,0x15  //Scancode = $75          KP8     CEDRIC - move cursor up
,0x1B  //Scancode = $76          Escape
,0x00  //Scancode = $77
,0x18  //Scancode = $78          F11     CEDRIC - delete line
,'+'   //Scancode = $79          KP+
,0x06  //Scancode = $7A          KP3     CEDRIC - move one line forward
,'-'   //Scancode = $7B          KP-
,'*'   //Scancode = $7C          KP*
,0x02  //Scancode = $7D          KP9     CEDRIC - line back
,0x00  //Scancode = $7E
,0x00  //Scancode = $7F
```

```
//Third dataset, shift = 1, shiftlock = 0
 ,0x00   //Scancode = $00
 ,0x05   //Scancode = $01          F9       CEDRIC - delete char
 ,0x00   //Scancode = $02
 ,0x19   //Scancode = $03          F5       CEDRIC - cut
 ,0x0E   //Scancode = $04          F3       CEDRIC - search and replace
 ,0x03   //Scancode = $05          F1       CEDRIC - search forward
 ,0x1A   //Scancode = $06          F2       CEDRIC - search back
 ,0x08   //Scancode = $07          F12      CEDRIC - delete left (backspace)
 ,0x00   //Scancode = $08
 ,0x17   //Scancode = $09          F10      CEDRIC - delete word
 ,0x13   //Scancode = $0A          F8       CEDRIC -
 ,0x10   //Scancode = $0B          F6       CEDRIC - paste
 ,0x01   //Scancode = $0C          F4       CEDRIC - global replace
 ,0x09   //Scancode = $0D          Tab
 ,'~'    //Scancode = $0E          ~
 ,0x00   //Scancode = $0F

 ,0x00   //Scancode = $10
 ,0x00   //Scancode = $11
 ,0x00   //Scancode = $12
 ,0x00   //Scancode = $13
 ,0x00   //Scancode = $14
 ,'Q'    //Scancode = $15          Q
 ,'!'    //Scancode = $16          !
 ,0x00   //Scancode = $17
 ,0x00   //Scancode = $18
 ,0x00   //Scancode = $19
 ,'Z'    //Scancode = $1A          Z
 ,'S'    //Scancode = $1B          S
 ,'A'    //Scancode = $1C          A
 ,'W'    //Scancode = $1D          W
 ,'@'    //Scancode = $1E          @
 ,0x00   //Scancode = $1F

 ,0x00   //Scancode = $20
 ,'C'    //Scancode = $21          C
 ,'X'    //Scancode = $22          X
 ,'D'    //Scancode = $23          D
 ,'E'    //Scancode = $24          E
 ,'$'    //Scancode = $25          $
 ,'#'    //Scancode = $26          #
 ,0x00   //Scancode = $27
 ,0x00   //Scancode = $28
 ,' '    //Scancode = $29          Space
 ,'V'    //Scancode = $2A          V
 ,'F'    //Scancode = $2B          F
 ,'T'    //Scancode = $2C          T
 ,'R'    //Scancode = $2D          R
 ,'%'    //Scancode = $2E          %
 ,0x00   //Scancode = $2F

 ,0x00   //Scancode = $30
 ,'N'    //Scancode = $31          N
 ,'B'    //Scancode = $32          B
 ,'H'    //Scancode = $33          H
 ,'G'    //Scancode = $34          G
 ,'Y'    //Scancode = $35          Y
 ,'^'    //Scancode = $36          ^
 ,0x00   //Scancode = $37
 ,0x00   //Scancode = $38
 ,0x00   //Scancode = $39
 ,'M'    //Scancode = $3A          M
 ,'J'    //Scancode = $3B          J
 ,'U'    //Scancode = $3C          U
 ,'&'    //Scancode = $3D          &
 ,'*'    //Scancode = $3E          *
 ,0x00   //Scancode = $3F
```

```
,0x00  //Scancode = $40
,'<'   //Scancode = $41          <
,'K'   //Scancode = $42          K
,'I'   //Scancode = $43          I
,'O'   //Scancode = $44          O
,')'   //Scancode = $45          )
,'('   //Scancode = $46          (
,0x00  //Scancode = $47
,0x00  //Scancode = $48
,'>'   //Scancode = $49          >
,'?'   //Scancode = $4A          ?
,'L'   //Scancode = $4B          L
,':'   //Scancode = $4C          :
,'P'   //Scancode = $4D          P
,'_'   //Scancode = $4E          _
,0x00  //Scancode = $4F

,0x00  //Scancode = $50
,0x00  //Scancode = $51
,'"'   //Scancode = $52          "
,0x00  //Scancode = $53
,'{'   //Scancode = $54          {
,'+'   //Scancode = $55          +
,0x00  //Scancode = $56
,0x00  //Scancode = $57
,0x00  //Scancode = $58
,0x00  //Scancode = $59
,0x0D  //Scancode = $5A          Enter
,'}'   //Scancode = $5B          }
,0x00  //Scancode = $5C
,'|'   //Scancode = $5D          |
,0x00  //Scancode = $5E
,0x00  //Scancode = $5F

,0x00  //Scancode = $60
,0x00  //Scancode = $61
,0x00  //Scancode = $62
,0x00  //Scancode = $63
,0x00  //Scancode = $64
,0x00  //Scancode = $65
,0x08  //Scancode = $66          Backspace
,0x00  //Scancode = $67
,0x00  //Scancode = $68
,0x0A  //Scancode = $69          KP1     CEDRIC - move to line end
,0x00  //Scancode = $6A
,0x0C  //Scancode = $6B          KP4     CEDRIC - move cursor left
,0x09  //Scancode = $6C          KP7     CEDRIC - (Tab)
,0x00  //Scancode = $6D
,0x00  //Scancode = $6E
,0x00  //Scancode = $6F

,0x00  //Scancode = $70          KP0
,'.'   //Scancode = $71          KP.
,0x04  //Scancode = $72          KP2     CEDRIC - move cursor down
,0x1C  //Scancode = $73          KP5     CEDRIC - point here
,0x12  //Scancode = $74          KP6     CEDRIC - move cursor right
,0x15  //Scancode = $75          KP8     CEDRIC - move cursor up
,0x1B  //Scancode = $76          Escape
,0x00  //Scancode = $77
,0x18  //Scancode = $78          F11     CEDRIC - delete line
,'+'   //Scancode = $79          KP+
,0x06  //Scancode = $7A          KP3     CEDRIC - move one line forward
,'-'   //Scancode = $7B          KP-
,'*'   //Scancode = $7C          KP*
,0x02  //Scancode = $7D          KP9     CEDRIC - line back
,0x00  //Scancode = $7E
,0x00  //Scancode = $7F
```

```
//Fourth dataset, shift = 1, shiftlock = 1
,0x00   //Scancode = $00
,0x05   //Scancode = $01          F9       CEDRIC - delete char
,0x00   //Scancode = $02
,0x19   //Scancode = $03          F5       CEDRIC - cut
,0x0E   //Scancode = $04          F3       CEDRIC - search and replace
,0x03   //Scancode = $05          F1       CEDRIC - search forward
,0x1A   //Scancode = $06          F2       CEDRIC - search back
,0x08   //Scancode = $07          F12      CEDRIC - delete left (backspace)
,0x00   //Scancode = $08
,0x17   //Scancode = $09          F10      CEDRIC - delete word
,0x13   //Scancode = $0A          F8       CEDRIC -
,0x10   //Scancode = $0B          F6       CEDRIC - paste
,0x01   //Scancode = $0C          F4       CEDRIC - global replace
,0x09   //Scancode = $0D          Tab
,'~'    //Scancode = $0E          ~
,0x00   //Scancode = $0F

,0x00   //Scancode = $10
,0x00   //Scancode = $11
,0x00   //Scancode = $12
,0x00   //Scancode = $13
,0x00   //Scancode = $14
,'q'    //Scancode = $15          q
,'!'    //Scancode = $16          !
,0x00   //Scancode = $17
,0x00   //Scancode = $18
,0x00   //Scancode = $19
,'z'    //Scancode = $1A          z
,'s'    //Scancode = $1B          s
,'a'    //Scancode = $1C          a
,'w'    //Scancode = $1D          w
,'@'    //Scancode = $1E          @
,0x00   //Scancode = $1F

,0x00   //Scancode = $20
,'c'    //Scancode = $21          c
,'x'    //Scancode = $22          x
,'d'    //Scancode = $23          d
,'e'    //Scancode = $24          e
,'$'    //Scancode = $25          $
,'#'    //Scancode = $26          #
,0x00   //Scancode = $27
,0x00   //Scancode = $28
,' '    //Scancode = $29          Space
,'v'    //Scancode = $2A          v
,'f'    //Scancode = $2B          f
,'t'    //Scancode = $2C          t
,'r'    //Scancode = $2D          r
,'%'    //Scancode = $2E          %
,0x00   //Scancode = $2F

,0x00   //Scancode = $30
,'n'    //Scancode = $31          n
,'b'    //Scancode = $32          b
,'h'    //Scancode = $33          h
,'g'    //Scancode = $34          g
,'y'    //Scancode = $35          y
,'^'    //Scancode = $36          ^
,0x00   //Scancode = $37
,0x00   //Scancode = $38
,0x00   //Scancode = $39
,'m'    //Scancode = $3A          m
,'j'    //Scancode = $3B          j
,'u'    //Scancode = $3C          u
,'&'    //Scancode = $3D          &
,'*'    //Scancode = $3E          *
,0x00   //Scancode = $3F
```

```
 ,0x00  //Scancode = $40
 ,'<'   //Scancode = $41          <
 ,'k'   //Scancode = $42          k
 ,'i'   //Scancode = $43          i
 ,'O'   //Scancode = $44          o
 ,')'   //Scancode = $45          )
 ,'('   //Scancode = $46          (
 ,0x00  //Scancode = $47
 ,0x00  //Scancode = $48
 ,'>'   //Scancode = $49          >
 ,'?'   //Scancode = $4A          ?
 ,'l'   //Scancode = $4B          l
 ,':'   //Scancode = $4C          :
 ,'p'   //Scancode = $4D          p
 ,'_'   //Scancode = $4E          _
 ,0x00  //Scancode = $4F

 ,0x00  //Scancode = $50
 ,0x00  //Scancode = $51
 ,'"'   //Scancode = $52          "
 ,0x00  //Scancode = $53
 ,'{'   //Scancode = $54          {
 ,'+'   //Scancode = $55          +
 ,0x00  //Scancode = $56
 ,0x00  //Scancode = $57
 ,0x00  //Scancode = $58
 ,0x00  //Scancode = $59
 ,0x0D  //Scancode = $5A          Enter
 ,'}'   //Scancode = $5B          }
 ,0x00  //Scancode = $5C
 ,'|'   //Scancode = $5D          |
 ,0x00  //Scancode = $5E
 ,0x00  //Scancode = $5F

 ,0x00  //Scancode = $60
 ,0x00  //Scancode = $61
 ,0x00  //Scancode = $62
 ,0x00  //Scancode = $63
 ,0x00  //Scancode = $64
 ,0x00  //Scancode = $65
 ,0x08  //Scancode = $66          Backspace
 ,0x00  //Scancode = $67
 ,0x00  //Scancode = $68
 ,0x0A  //Scancode = $69          KP1     CEDRIC - move to line end
 ,0x00  //Scancode = $6A
 ,0x0C  //Scancode = $6B          KP4     CEDRIC - move cursor left
 ,0x09  //Scancode = $6C          KP7     CEDRIC - (Tab)
 ,0x00  //Scancode = $6D
 ,0x00  //Scancode = $6E
 ,0x00  //Scancode = $6F

 ,0x00  //Scancode = $70          KP0
 ,'.'   //Scancode = $71          KP.
 ,0x04  //Scancode = $72          KP2     CEDRIC - move cursor down
 ,0x1C  //Scancode = $73          KP5     CEDRIC - point here
 ,0x12  //Scancode = $74          KP6     CEDRIC - move cursor right
 ,0x15  //Scancode = $75          KP8     CEDRIC - move cursor up
 ,0x1B  //Scancode = $76          Escape
 ,0x00  //Scancode = $77
 ,0x18  //Scancode = $78          F11     CEDRIC - delete line
 ,'+'   //Scancode = $79          KP+
 ,0x06  //Scancode = $7A          KP3     CEDRIC - move one line forward
 ,'-'   //Scancode = $7B          KP-
 ,'*'   //Scancode = $7C          KP*
 ,0x02  //Scancode = $7D          KP9     CEDRIC - line back
 ,0x00  //Scancode = $7E
 ,0x00  //Scancode = $7F
};
```

Appendix 4 - Graphics display codes

The graphics display code provides a simple way to generate pictures using the internal graphics drivers. Here is an example display list:-

```
OPT NOL
LIB GRAPHICS.MAC
OPT LIS
*
CLEAR_SCREEN
SET_PEN_TYPE 0,$FFFF
MOVE_CURSOR 100,100
PLOT_LINE 200,200
PLOT_TEXT 'HI THERE!'
END_DRAW
*
END
```

This list should be assembled with ASMB in the normal manner, placed into memory using GET, and then the FLEX PLAY command should be used to draw the picture.

The graphics display codes are three byte 'opcodes' defined the macro set GRAPHICS.MAC. The available commands are :-

NULL
Do nothing.

CLEAR_SCREEN
Clear the graphics screen.

MOVE_CURSOR x-coord,y-coord
Moves the cursor to the given coords.

PLOT_POINT x-coord,y-coord
Plots a point at the given coords.

PLOT_LINE x-coord,y-coord
Plots a line from the present cursor position to the given coords.

PLOT_RECTANGE sidex,sidey
Plots a rectangle (bottom rh corner is present coords), with given sides.

PLOT_CIRCLE radius
Plots a circle (centre is present coords) with given radius where (0<radius<127).

PLOT_TEXT 'text string'
Plots the text string from the given coords.

SET_PEN_TYPE pen_type,profile
Sets the pen type and drawing profile.

SET_TEXT_ZOOM zoom_factor
Sets the text size (0<zoom_factor<15)

END_DRAW
Ends the drawing process.

Appendix 5 - Default display character set

```
// this array contains the CHARS1.CHR character set
    char charSet[1086] = {
// character set attributes
    84, // screen chars/row
    24, // screen rows
    7,  // char x pixels
    11, // char y rows
    9,  // bounding box pixels
    12, // bounding box rows
    4,  // initial attributes
    11  // char spacing (pixels)

    // character set patterns
    , 0b00000000 //Character = ' '
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
   //
    , 0b00001000 //Character = '!'
    , 0b00001000
    , 0b00001000
    , 0b00001000
    , 0b00001000
    , 0b00000000
    , 0b00001000
    , 0b00001000
    , 0b00000000
    , 0b00000000
    , 0b00000000
   //
    , 0b00100100 //Character = '"'
    , 0b00100100
    , 0b00100100
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
   //
    , 0b00010100 //Character = '#'
    , 0b00010100
    , 0b00010100
    , 0b01111111
    , 0b00010100
    , 0b01111111
    , 0b00010100
    , 0b00010100
    , 0b00010100
    , 0b00000000
    , 0b00000000
```

```
  //
   , 0b00001000 //Character = '$'
   , 0b00111111
   , 0b01001000
   , 0b01001000
   , 0b00111110
   , 0b00001001
   , 0b00001001
   , 0b01111110
   , 0b00001000
   , 0b00000000
   , 0b00000000
  //
   , 0b00100000 //Character = '%'
   , 0b01010001
   , 0b00100010
   , 0b00000100
   , 0b00001000
   , 0b00010000
   , 0b00100010
   , 0b01000101
   , 0b00000010
   , 0b00000000
   , 0b00000000
  //
   , 0b00111000 //Character = '&'
   , 0b01000100
   , 0b01000100
   , 0b00101000
   , 0b00010000
   , 0b00101001
   , 0b01000110
   , 0b01000110
   , 0b00111001
   , 0b00000000
   , 0b00000000
  //
   , 0b00001100 //Character = '''
   , 0b00001100
   , 0b00001000
   , 0b00010000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
  //
   , 0b00000100 //Character = '('
   , 0b00001000
   , 0b00010000
   , 0b00010000
   , 0b00010000
   , 0b00010000
   , 0b00010000
   , 0b00001000
   , 0b00000100
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b00010000 //Character = ')'
   , 0b00001000
   , 0b00000100
   , 0b00000100
   , 0b00000100
   , 0b00000100
   , 0b00000100
   , 0b00001000
   , 0b00010000
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = '//'
   , 0b00001000
   , 0b01001001
   , 0b00101010
   , 0b01111111
   , 0b00101010
   , 0b01001001
   , 0b00001000
   , 0b00000000
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = '+'
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b01111111
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00000000
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = ','
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00011000
   , 0b00011000
   , 0b00010000
   , 0b00100000
  //
   , 0b00000000 //Character = '-'
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00111110
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b00000000 //Character = 'fred'
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
   , 0b00011000
   , 0b00011000
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = '/'
   , 0b00000001
   , 0b00000010
   , 0b00000100
   , 0b00001000
   , 0b00010000
   , 0b00100000
   , 0b01000000
   , 0b00000000
   , 0b00000000
   , 0b00000000
  //
   , 0b00111110 //Character = '0'
   , 0b01000001
   , 0b01000011
   , 0b01000101
   , 0b01001001
   , 0b01010001
   , 0b01100001
   , 0b01000001
   , 0b00111110
   , 0b00000000
   , 0b00000000
  //
   , 0b00001000 //Character = '1'
   , 0b00011000
   , 0b00101000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00111110
   , 0b00000000
   , 0b00000000
  //
   , 0b00111110 //Character = '2'
   , 0b01000001
   , 0b00000001
   , 0b00000010
   , 0b00011100
   , 0b00100000
   , 0b01000000
   , 0b01000000
   , 0b01111111
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b00111110 //Character = '3'
   , 0b01000001
   , 0b00000001
   , 0b00000001
   , 0b00011110
   , 0b00000001
   , 0b00000001
   , 0b01000001
   , 0b00111110
   , 0b00000000
   , 0b00000000
  //
   , 0b00000010 //Character = '4'
   , 0b00000110
   , 0b00001010
   , 0b00010010
   , 0b00100010
   , 0b01000010
   , 0b01111111
   , 0b00000010
   , 0b00000010
   , 0b00000000
   , 0b00000000
  //
   , 0b01111111 //Character = '5'
   , 0b01000000
   , 0b01000000
   , 0b01111100
   , 0b00000010
   , 0b00000001
   , 0b00000001
   , 0b01000010
   , 0b00111100
   , 0b00000000
   , 0b00000000
  //
   , 0b00011110 //Character = '6'
   , 0b00100000
   , 0b01000000
   , 0b01000000
   , 0b01111110
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00111110
   , 0b00000000
   , 0b00000000
  //
   , 0b01111111 //Character = '7'
   , 0b01000001
   , 0b00000010
   , 0b00000100
   , 0b00001000
   , 0b00010000
   , 0b00010000
   , 0b00010000
   , 0b00010000
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b00111110 //Character = '8'
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00111110
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00111110
   , 0b00000000
   , 0b00000000
  //
   , 0b00111110 //Character = '9'
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00111111
   , 0b00000001
   , 0b00000001
   , 0b00000010
   , 0b00111100
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = ':'
   , 0b00000000
   , 0b00000000
   , 0b00011000
   , 0b00011000
   , 0b00000000
   , 0b00000000
   , 0b00011000
   , 0b00011000
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = ';'
   , 0b00000000
   , 0b00000000
   , 0b00011000
   , 0b00011000
   , 0b00000000
   , 0b00000000
   , 0b00011000
   , 0b00011000
   , 0b00010000
   , 0b00100000
  //
   , 0b00000100 //Character = '<'
   , 0b00001000
   , 0b00010000
   , 0b00100000
   , 0b01000000
   , 0b00100000
   , 0b00010000
   , 0b00001000
   , 0b00000100
   , 0b00000000
   , 0b00000000
```

```
  //
    , 0b00000000 //Character = '='
    , 0b00000000
    , 0b00000000
    , 0b00111110
    , 0b00000000
    , 0b00111110
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
   //
    , 0b00010000 //Character = '>'
    , 0b00001000
    , 0b00000100
    , 0b00000010
    , 0b00000001
    , 0b00000010
    , 0b00000100
    , 0b00001000
    , 0b00010000
    , 0b00000000
    , 0b00000000
   //
    , 0b00011110 //Character = '?'
    , 0b00100001
    , 0b00100001
    , 0b00000001
    , 0b00000110
    , 0b00001000
    , 0b00001000
    , 0b00000000
    , 0b00001000
    , 0b00000000
    , 0b00000000
   //
    , 0b00011110 //Character = '@'
    , 0b00100001
    , 0b01001101
    , 0b01010101
    , 0b01010101
    , 0b01011110
    , 0b01000000
    , 0b00100000
    , 0b00011110
    , 0b00000000
    , 0b00000000
   //
    , 0b00011100 //Character = 'A'
    , 0b00100010
    , 0b01000001
    , 0b01000001
    , 0b01000001
    , 0b01111111
    , 0b01000001
    , 0b01000001
    , 0b01000001
    , 0b00000000
    , 0b00000000
```

```
  //
    , 0b01111110 //Character = 'B'
    , 0b00100001
    , 0b00100001
    , 0b00100001
    , 0b00111110
    , 0b00100001
    , 0b00100001
    , 0b00100001
    , 0b01111110
    , 0b00000000
    , 0b00000000
   //
    , 0b00011110 //Character = 'C'
    , 0b00100001
    , 0b01000000
    , 0b01000000
    , 0b01000000
    , 0b01000000
    , 0b01000000
    , 0b00100001
    , 0b00011110
    , 0b00000000
    , 0b00000000
   //
    , 0b01111100 //Character = 'D'
    , 0b01000010
    , 0b01000001
    , 0b01000001
    , 0b01000001
    , 0b01000001
    , 0b01000010
    , 0b01111100
    , 0b00000000
    , 0b00000000
   //
    , 0b01111111 //Character = 'E'
    , 0b01000000
    , 0b01000000
    , 0b01000000
    , 0b01111000
    , 0b01000000
    , 0b01000000
    , 0b01000000
    , 0b01111111
    , 0b00000000
    , 0b00000000
   //
    , 0b01111111 //Character = 'F'
    , 0b01000000
    , 0b01000000
    , 0b01000000
    , 0b01111000
    , 0b01000000
    , 0b01000000
    , 0b01000000
    , 0b01000000
    , 0b00000000
    , 0b00000000
```

```
  //
   , 0b00011110 //Character = 'G'
   , 0b00100001
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b01001111
   , 0b01000001
   , 0b00100001
   , 0b00011110
   , 0b00000000
   , 0b00000000
  //
   , 0b01000001 //Character = 'H'
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01111111
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00000000
   , 0b00000000
  //
   , 0b00111110 //Character = 'I'
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00111110
   , 0b00000000
   , 0b00000000
  //
   , 0b00011111 //Character = 'J'
   , 0b00000100
   , 0b00000100
   , 0b00000100
   , 0b00000100
   , 0b00000100
   , 0b00000100
   , 0b01000100
   , 0b00111000
   , 0b00000000
   , 0b00000000
  //
   , 0b01000001 //Character = 'K'
   , 0b01000010
   , 0b01000100
   , 0b01001000
   , 0b01010000
   , 0b01101000
   , 0b01000100
   , 0b01000010
   , 0b01000001
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b01000000 //Character = 'L'
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b01111111
   , 0b00000000
   , 0b00000000
  //
   , 0b01000001 //Character = 'M'
   , 0b01100011
   , 0b01010101
   , 0b01001001
   , 0b01001001
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00000000
   , 0b00000000
  //
   , 0b01000001 //Character = 'N'
   , 0b01100001
   , 0b01010001
   , 0b01001001
   , 0b01000101
   , 0b01000011
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00000000
   , 0b00000000
  //
   , 0b00011100 //Character = 'O'
   , 0b00100010
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00100010
   , 0b00011100
   , 0b00000000
   , 0b00000000
  //
   , 0b01111110 //Character = 'P'
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01111110
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b00011100 //Character = 'Q'
   , 0b00100010
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01001001
   , 0b01000101
   , 0b00100010
   , 0b00011101
   , 0b00000000
   , 0b00000000
  //
   , 0b01111110 //Character = 'R'
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01111110
   , 0b01001000
   , 0b01000100
   , 0b01000010
   , 0b01000001
   , 0b00000000
   , 0b00000000
  //
   , 0b00111110 //Character = 'S'
   , 0b01000001
   , 0b01000000
   , 0b01000000
   , 0b00111110
   , 0b00000001
   , 0b00000001
   , 0b01000001
   , 0b00111110
   , 0b00000000
   , 0b00000000
  //
   , 0b01111111 //Character = 'T'
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00000000
   , 0b00000000
  //
   , 0b01000001 //Character = 'U'
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00111110
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b01000001 //Character = 'V'
   , 0b01000001
   , 0b01000001
   , 0b00100010
   , 0b00100010
   , 0b00010100
   , 0b00010100
   , 0b00001000
   , 0b00001000
   , 0b00000000
   , 0b00000000
  //
   , 0b01000001 //Character = 'W'
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01001001
   , 0b01001001
   , 0b01010101
   , 0b01100011
   , 0b01000001
   , 0b00000000
   , 0b00000000
  //
   , 0b01000001 //Character = 'X'
   , 0b01000001
   , 0b00100010
   , 0b00010100
   , 0b00001000
   , 0b00010100
   , 0b00100010
   , 0b01000001
   , 0b01000001
   , 0b00000000
   , 0b00000000
  //
   , 0b01000001 //Character = 'Y'
   , 0b01000001
   , 0b00100010
   , 0b00010100
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00000000
   , 0b00000000
  //
   , 0b01111111 //Character = 'Z'
   , 0b00000001
   , 0b00000010
   , 0b00000100
   , 0b00001000
   , 0b00010000
   , 0b00100000
   , 0b01000000
   , 0b01111111
   , 0b00000000
   , 0b00000000
```

```
  //
    , 0b00111110 //Character = '['
    , 0b00100000
    , 0b00100000
    , 0b00100000
    , 0b00100000
    , 0b00100000
    , 0b00100000
    , 0b00100000
    , 0b00111110
    , 0b00000000
    , 0b00000000
   //
    , 0b00000000 //Character = '\'
    , 0b01000000
    , 0b00100000
    , 0b00010000
    , 0b00001000
    , 0b00000100
    , 0b00000010
    , 0b00000001
    , 0b00000000
    , 0b00000000
    , 0b00000000
   //
    , 0b00111110 //Character = ']'
    , 0b00000010
    , 0b00000010
    , 0b00000010
    , 0b00000010
    , 0b00000010
    , 0b00000010
    , 0b00000010
    , 0b00111110
    , 0b00000000
    , 0b00000000
   //
    , 0b00001000 //Character = '^'
    , 0b00011100
    , 0b00101010
    , 0b01001001
    , 0b00001000
    , 0b00001000
    , 0b00001000
    , 0b00001000
    , 0b00001000
    , 0b00000000
    , 0b00000000
   //
    , 0b00000000 //Character = '_'
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00111110
    , 0b00000000
    , 0b00000000
```

```
  //
    , 0b00011000 //Character = '`'
    , 0b00011000
    , 0b00001000
    , 0b00000100
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
   //
    , 0b00000000 //Character = 'a'
    , 0b00000000
    , 0b00000000
    , 0b00111100
    , 0b00000010
    , 0b00111110
    , 0b01000010
    , 0b01000010
    , 0b00111101
    , 0b00000000
    , 0b00000000
   //
    , 0b01000000 //Character = 'b'
    , 0b01000000
    , 0b01000000
    , 0b01011100
    , 0b01100010
    , 0b01000010
    , 0b01000010
    , 0b01100010
    , 0b01011100
    , 0b00000000
    , 0b00000000
   //
    , 0b00000000 //Character = 'c'
    , 0b00000000
    , 0b00000000
    , 0b00111100
    , 0b01000010
    , 0b01000000
    , 0b01000000
    , 0b01000010
    , 0b00111100
    , 0b00000000
    , 0b00000000
   //
    , 0b00000010 //Character = 'd'
    , 0b00000010
    , 0b00000010
    , 0b00111010
    , 0b01000110
    , 0b01000010
    , 0b01000010
    , 0b01000110
    , 0b00111010
    , 0b00000000
    , 0b00000000
```

```
  //
   , 0b00000000 //Character = 'e'
   , 0b00000000
   , 0b00000000
   , 0b00111100
   , 0b01000010
   , 0b01111110
   , 0b01000000
   , 0b01000000
   , 0b00111100
   , 0b00000000
   , 0b00000000
  //
   , 0b00001100 //Character = 'f'
   , 0b00010010
   , 0b00010000
   , 0b00010000
   , 0b01111100
   , 0b00010000
   , 0b00010000
   , 0b00010000
   , 0b00010000
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 'g'
   , 0b00000000
   , 0b00000000
   , 0b00111010
   , 0b01000110
   , 0b01000010
   , 0b01000110
   , 0b00111010
   , 0b00000010
   , 0b01000010
   , 0b00111100
  //
   , 0b01000000 //Character = 'h'
   , 0b01000000
   , 0b01000000
   , 0b01011100
   , 0b01100010
   , 0b01000010
   , 0b01000010
   , 0b01000010
   , 0b01000010
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 'i'
   , 0b00001000
   , 0b00000000
   , 0b00011000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00011100
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b00000000 //Character = 'j'
   , 0b00000010
   , 0b00000000
   , 0b00000110
   , 0b00000010
   , 0b00000010
   , 0b00000010
   , 0b00000010
   , 0b00000010
   , 0b00100010
   , 0b00011100
  //
   , 0b01000000 //Character = 'k'
   , 0b01000000
   , 0b01000000
   , 0b01000100
   , 0b01001000
   , 0b01010000
   , 0b01101000
   , 0b01000100
   , 0b01000010
   , 0b00000000
   , 0b00000000
  //
   , 0b00011000 //Character = 'l'
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00001000
   , 0b00011100
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 'm'
   , 0b00000000
   , 0b00000000
   , 0b01110110
   , 0b01001001
   , 0b01001001
   , 0b01001001
   , 0b01001001
   , 0b01001001
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 'n'
   , 0b00000000
   , 0b00000000
   , 0b01011100
   , 0b01100010
   , 0b01000010
   , 0b01000010
   , 0b01000010
   , 0b01000010
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b00000000 //Character = 'o'
   , 0b00000000
   , 0b00000000
   , 0b00111110
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00111110
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 'p'
   , 0b00000000
   , 0b00000000
   , 0b01011100
   , 0b01100010
   , 0b01000010
   , 0b01100010
   , 0b01011100
   , 0b01000000
   , 0b01000000
   , 0b01000000
  //
   , 0b00000000 //Character = 'q'
   , 0b00000000
   , 0b00000000
   , 0b00111010
   , 0b01000110
   , 0b01000010
   , 0b01000110
   , 0b00111010
   , 0b00000010
   , 0b00000010
   , 0b00000011
  //
   , 0b00000000 //Character = 'r'
   , 0b00000000
   , 0b00000000
   , 0b01011100
   , 0b01100010
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b01000000
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 's'
   , 0b00000000
   , 0b00000000
   , 0b00111100
   , 0b01000010
   , 0b00110000
   , 0b00001100
   , 0b01000010
   , 0b00111100
   , 0b00000000
   , 0b00000000
```

```
  //
   , 0b00000000 //Character = 't'
   , 0b00010000
   , 0b00010000
   , 0b01111100
   , 0b00010000
   , 0b00010000
   , 0b00010000
   , 0b00010010
   , 0b00001100
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 'u'
   , 0b00000000
   , 0b00000000
   , 0b01000010
   , 0b01000010
   , 0b01000010
   , 0b01000010
   , 0b01000110
   , 0b00111010
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 'v'
   , 0b00000000
   , 0b00000000
   , 0b01000001
   , 0b01000001
   , 0b01000001
   , 0b00100010
   , 0b00010100
   , 0b00001000
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 'w'
   , 0b00000000
   , 0b00000000
   , 0b01000001
   , 0b01001001
   , 0b01001001
   , 0b01001001
   , 0b01001001
   , 0b00110110
   , 0b00000000
   , 0b00000000
  //
   , 0b00000000 //Character = 'x'
   , 0b00000000
   , 0b00000000
   , 0b01000010
   , 0b00100100
   , 0b00011000
   , 0b00011000
   , 0b00100100
   , 0b01000010
   , 0b00000000
   , 0b00000000
```

```
  //
    , 0b00000000 //Character = 'y'
    , 0b00000000
    , 0b00000000
    , 0b01000010
    , 0b01000010
    , 0b01000010
    , 0b01000110
    , 0b00111010
    , 0b00000010
    , 0b01000010
    , 0b00111100
   //
    , 0b00000000 //Character = 'z'
    , 0b00000000
    , 0b00000000
    , 0b01111110
    , 0b00000100
    , 0b00001000
    , 0b00010000
    , 0b00100000
    , 0b01111110
    , 0b00000000
    , 0b00000000
   //
    , 0b00001100 //Character = '{'
    , 0b00010000
    , 0b00010000
    , 0b00010000
    , 0b00100000
    , 0b00010000
    , 0b00010000
    , 0b00010000
    , 0b00001100
    , 0b00000000
    , 0b00000000
   //
    , 0b00001000 //Character = '|'
    , 0b00001000
    , 0b00001000
    , 0b00000000
    , 0b00001000
    , 0b00001000
    , 0b00001000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
   //
    , 0b00011000 //Character = '}'
    , 0b00000100
    , 0b00000100
    , 0b00000100
    , 0b00000010
    , 0b00000100
    , 0b00000100
    , 0b00000100
    , 0b00011000
    , 0b00000000
    , 0b00000000
```

```
   //
    , 0b00110000 //Character = '~'
    , 0b01001001
    , 0b00000110
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b00000000
   //
    , 0b00101010 //Character = 'del'
    , 0b01010101
    , 0b00101010
    , 0b01010101
    , 0b00101010
    , 0b01010101
    , 0b00101010
    , 0b01010101
    , 0b00101010
    , 0b00000000
    , 0b00000000
   //
    , 0b01111111 // Cursor type 1
    , 0b01111111
    , 0b01111111
    , 0b01111111
    , 0b01111111
    , 0b01111111
    , 0b01111111
    , 0b01111111
    , 0b01111111
    , 0b00000000
    , 0b00000000
   //
    , 0b01100011 // Cursor type 2
    , 0b01000001
    , 0b01000001
    , 0b00000000
    , 0b00000000
    , 0b00000000
    , 0b01000001
    , 0b01000001
    , 0b01100011
    , 0b00000000
    , 0b00000000

};
```

Appendix 6 - MB2K2 promdisk contents

The promdisk is part of the Xmos boot image and is loaded into flash at the same time as the firmware image via the Xmos tools.

By default the promdisk is read only but can be set to read/write with the 'PDRW' command and write protected again with 'PDRO'. Although the promdisk can be written to, there is no wear levelling and repeated writes (1,000's) will eventually 'wear out' the part of the flash image containing the FLEX SIR and other frequently written sectors.

FLEX binaries

FLEX.COR - FLEX 3.01 binary used by the 'BF' command, replace to use different versions of FLEX.

ERRORS.SYS

STARTUP.TXT - Gets the time/date and sets TTYSET and ASN parameters from the battery backed RTC

Standard FLEX utilities

APPEND.CMD
ASN.CMD
BUILD.CMD
CAT.CMD
COPY.CMD
DATE.CMD
DELETE.CMD
ECHO.CMD
EXEC.CMD
I.CMD
JUMP.CMD
LIST.CMD
N.CMD
O.CMD
PROT.CMD
RENAME.CMD
SAVE.CMD
SAVETXT.CMD
TTYSET.CMD
TOUCH.CMD
VERIFY.CMD
VERSION.CMD
XOUT.CMD
Y.CMD

Additional FLEX utilities
CMPMEM.CMD
CONTIN.CMD
COMPARE.CMD
DATECOPY.CMD
DIR.CMD
DUMP.CMD
FILES.CMD
FIND.CMD
FREE.CMD
HECHO.CMD
MAP.CMD
MEMEND.CMD
PDEL.CMD
RUN.CMD
SPLIT.CMD
ZAP.CMD

FLEX Diagnostic utilities
EXAMINE.CMD
FILETEST.CMD
REBUILD.CMD
UNDELETE.CMD
VALIDATE.CMD

Text editors
E.CMD - This is the CEDRIC editor configured for the MB2K2's
PS/2 keyboard layout
STYLO.CMD - Stylograph configured for the MB2K2
STYHLP1.TXT
STYHLP2.TXT
STYHLP3.TXT
STYHLP4.TXT
STYHLP5.TXT
STYHLP6.TXT

TSC 6809 Assembler
ASMB.CMD

Dynamite 6809 Disassembler
DISA.CMD
DISLBL09.BIN

TSC Debug Package
DEBUG.CMD

Windrush PL/9 compiler

PL9.CMD
PL9_TD.CMD
SETPL9.CMD
HEXGLOBL.DEF
TRUFALSE.DEF
PL9.ERR
BASTRING.LIB
BITIO.LIB
REALCON.LIB
REALIO.LIB
SCIPACK.LIB
SORT.LIB
STRSUBS.LIB
TERMSUBS.LIB
SETPL9.PL9

James McCosh C compiler

CC.CMD
CPREP.CMD
CPASS1.CMD
CPASS2.CMD
COPT.CMD
CASM.CMD
CLOAD.CMD
CLIB.LIB
CSTART.R
CTYPE.H
FLEX.H
SETJMP.H
STDIO.H
HELLO.C

MB2/MB2K2 specific utilities

MONLINK.TXT - MON09 6809 assembly header file
FLEXLINK.TXT - FLEX 6809 assembly header file
ALLOCATE.CMD - allocate logical disk types to FLEX drive numbers
TIME.CMD - Time and date from the battery backed RTC
SETTIME.CMD - set the time and date in the battery backed RTC
GRAPH.CMD - switch display to graphics
TEXT.CMD - switch display to text
CLEARG.CMD - clear graphics screen
PRETTY.CMD - MB2 terminal emulator with higher resolution character set
FAST.CMD - MB2K2 terminal emulator using uPD7220A command extensions
NORMAL.CMD - revert to internal MON09 terminal emulator
SCOPY.CMD - fast track based disk copy (only for disks with the same format)
PDRW.CMD - enable writes to the promdisk
PDRO.CMD - write protect the promdisk
GRAPHICS.MAC - macro commands to define graphics vector list
PLAY.CMD - playback pre-defined vector list
INTERP.CMD - decode vector list to text

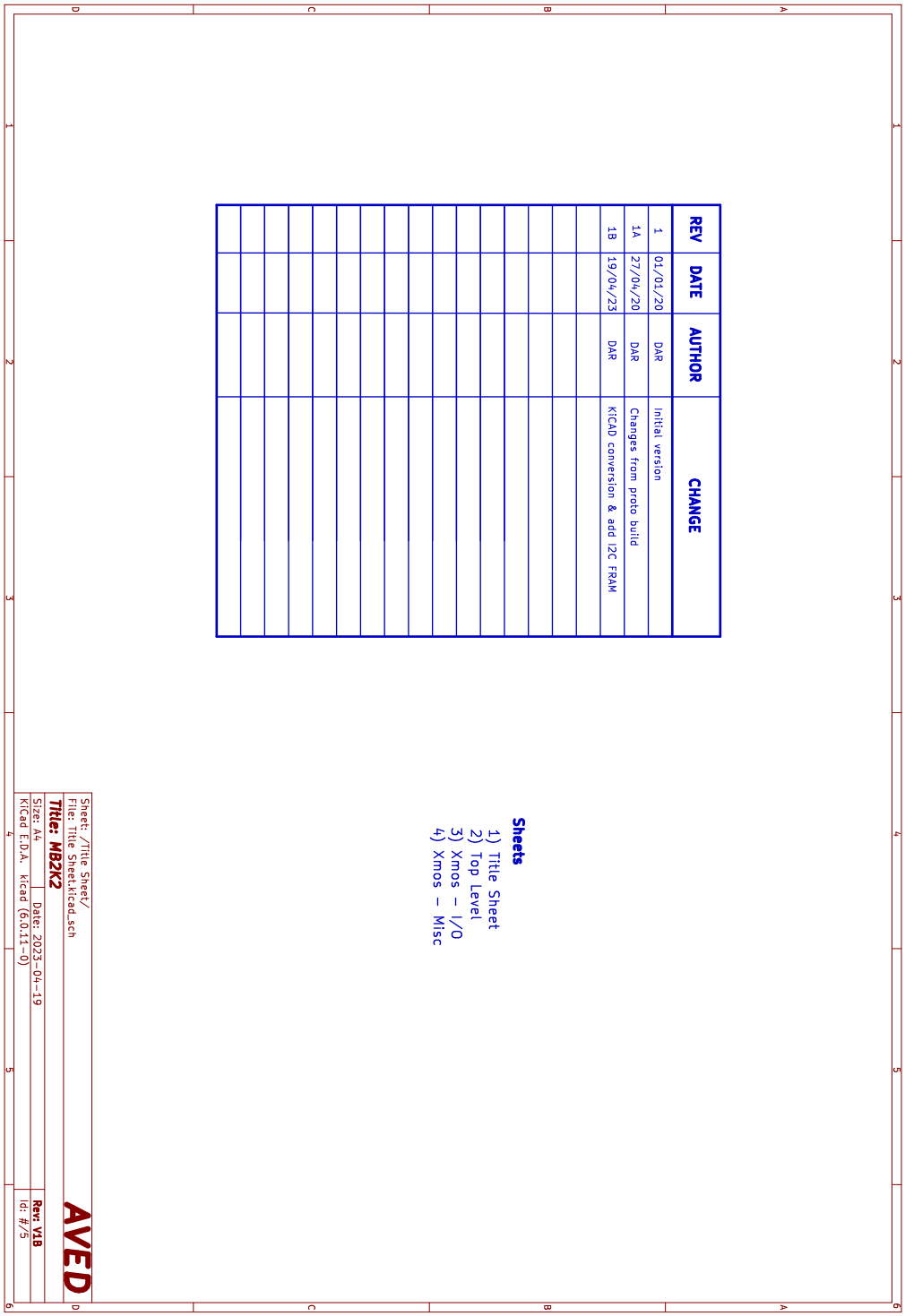
FLEXNet utilities

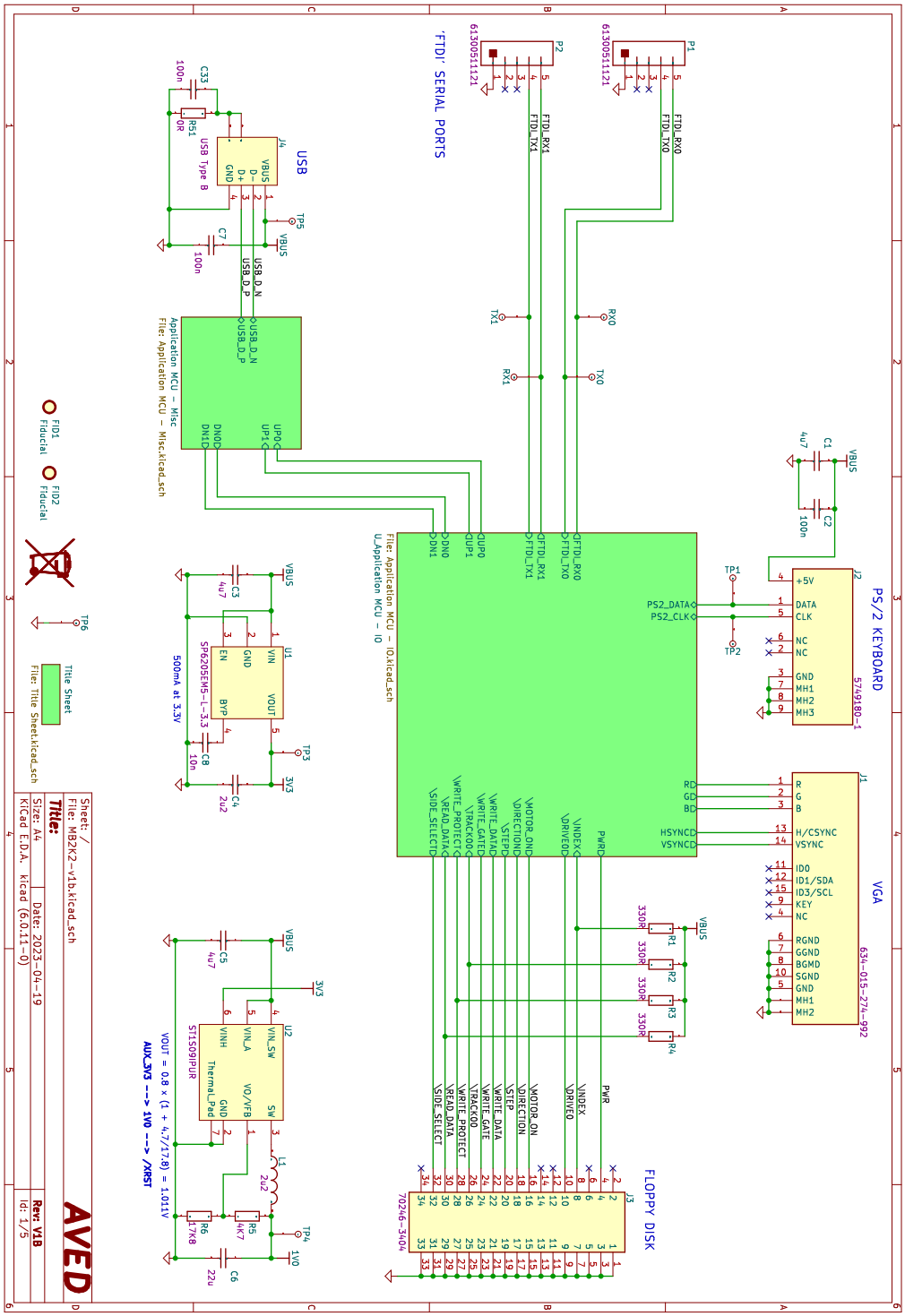
RMOUNT.CMD	- mount remote disk image, or show currently mounted images RMOUNT must be run before using any other FLEXNet command
RDRIVE.CMD	- show current remote directory
RLIST.CMD	- folder list of remote directory
RDIR.CMD	- file list of the remote directory
RCD.CMD	- change directory in the remote system
RCREATE.CMD	- create remote disk image
RDELETE.CMD	- delete remote disk image
RESYNC.CMD	- resynchronise the serial link to the remote system
REXIT.CMD	- close remote volumes and shut down connection

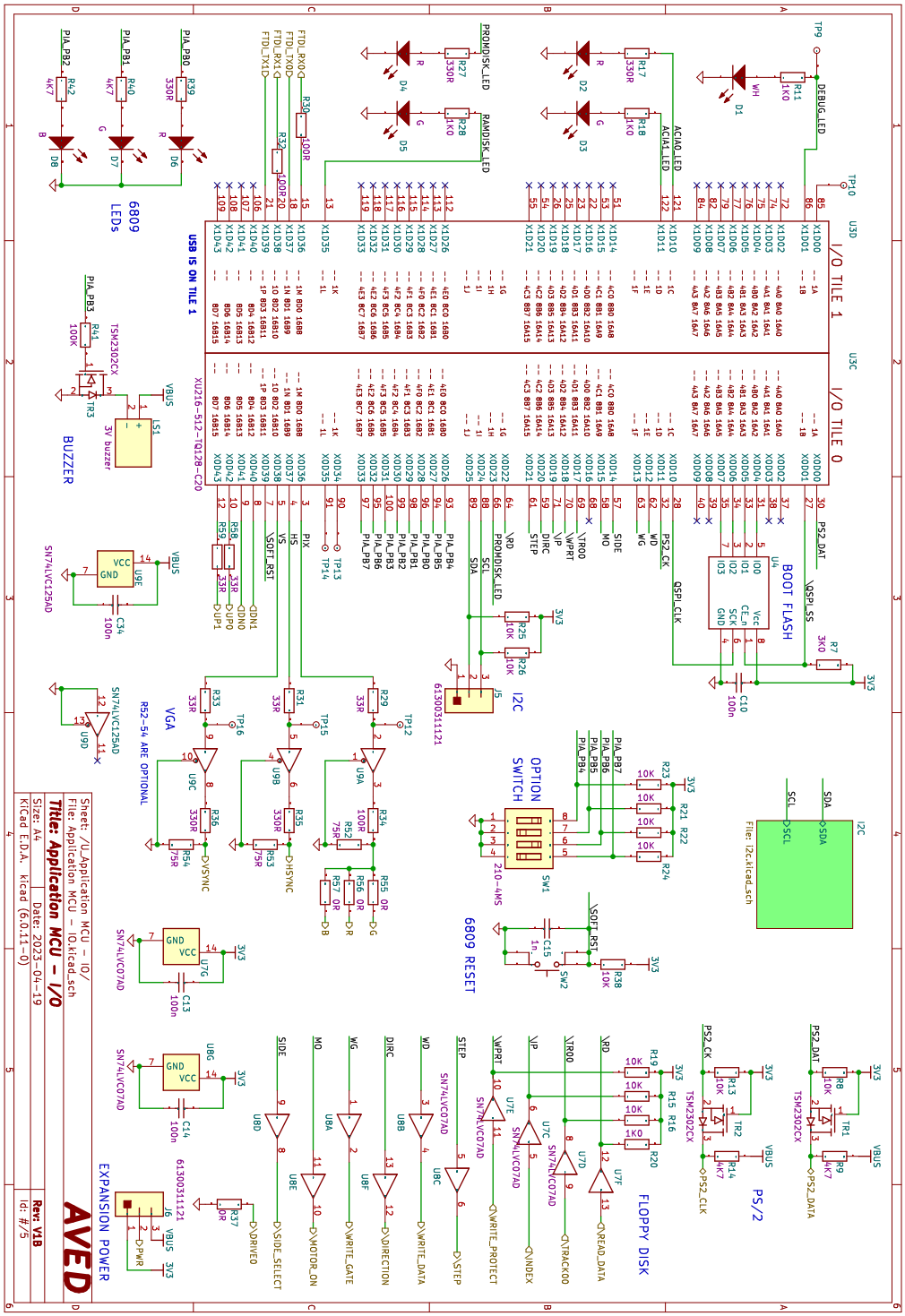
Misc

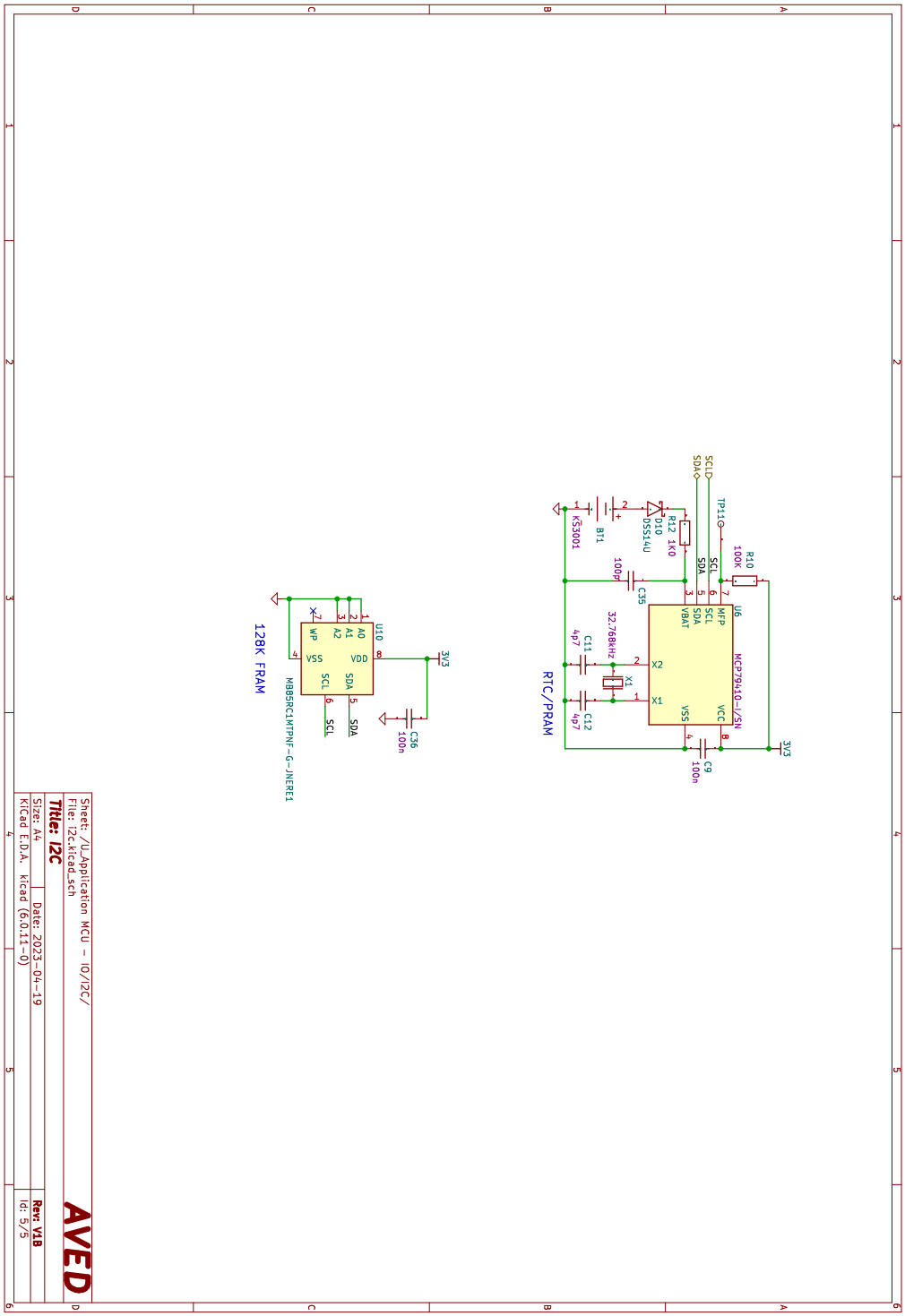
DEMO.CMD	- the MB2 graphics line drawing demo
ADV.CMD	- The classic Adventure game (overwrites FLEX due to large size)
ADVENTUR.SYS	- database for ADV.CMD
STARTREK.CMD	- A version of the text based Trek game
STARTREK.TXT	- source code for the above

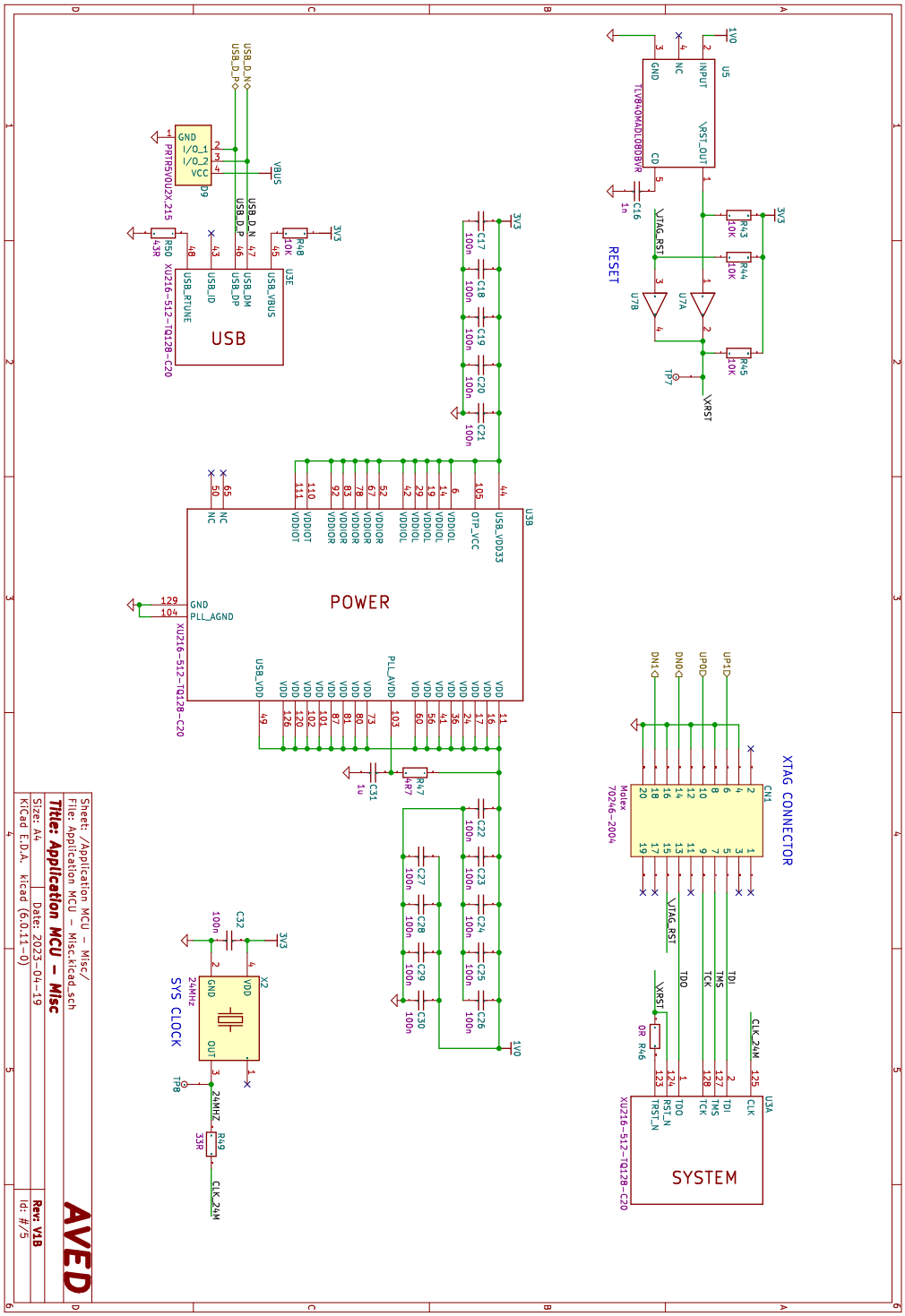
Appendix 7 - Schematics



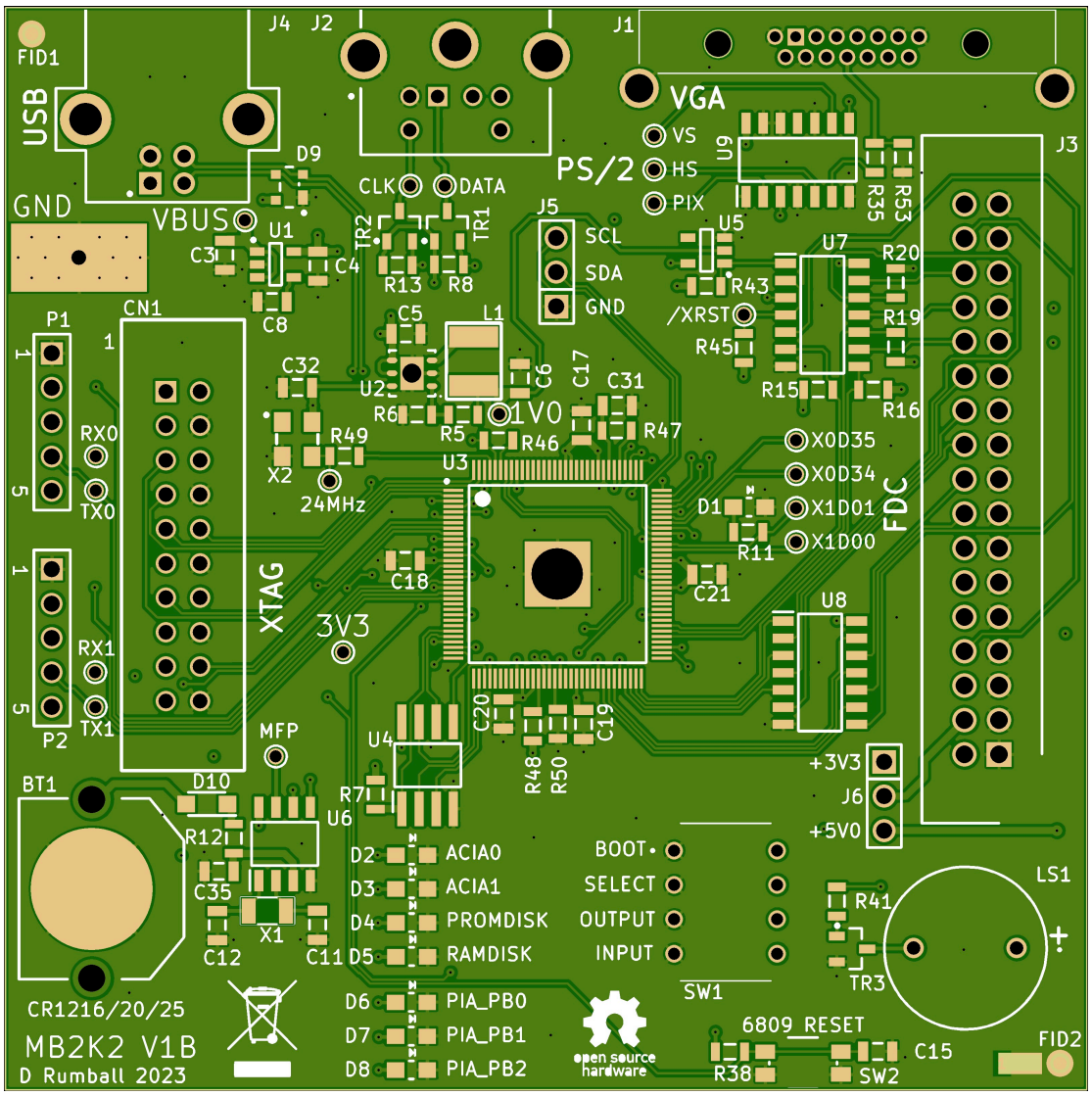

'FTDI' SERIAL PORTS
USB
PS/2 KEYBOARD
VGA
FLOPPY DISK
Application MCU - Misc
Application MCU - IO
Title Sheet
AVED
Rev: V1B
AUX_3V3 --> 1V0 --> /XRST


I/O TILE 1
I/O TILE 0
BOOT FLASH
I2C
OPTION SWITCH
6809 RESET
FLOPPY DISK
PS/2
VGA
BUZZER
6809 LEDs
EXPANSION POWER
USB IS ON TILE 1
Title: Application MCU – I/O
AVED
Rev V1B

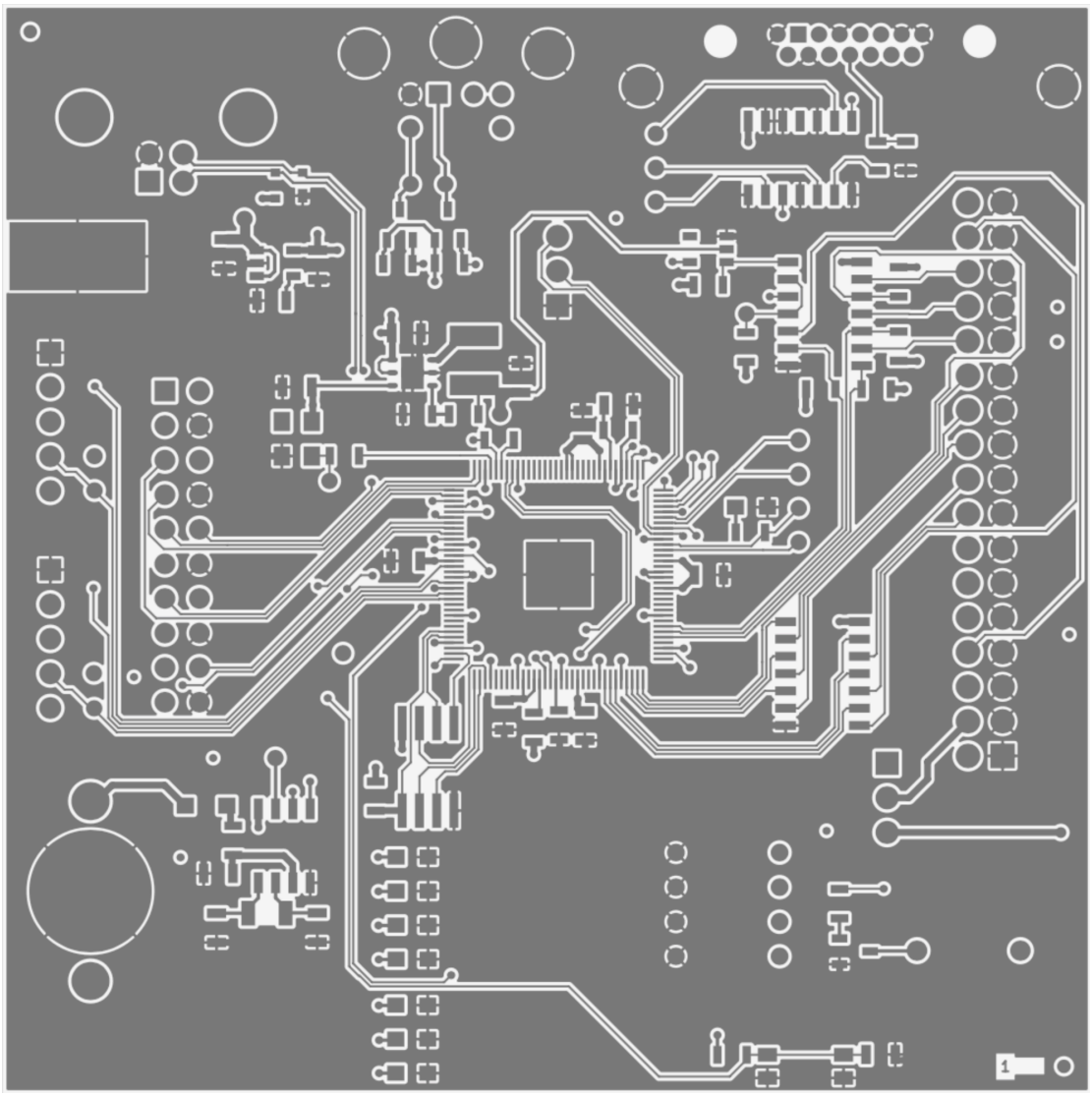

SCLD
SDAD
TP11
R10
100K
R12 1KO
D10
DSS14U
BT1
KS3001
100pF
C35
U6
MCP79410-I/SN
MFP
SCL
SDA
VBAT
VSS
VCC
X2
X1
32.768KHz
X1
C11
4p7
C12
4p7
C9
100n
3V3
RTC/PRAM
U10
M24M02-DRMN6TP
WP
A0
A1
A2
VSS
VDD
SCL
SDA
C36
100n
128K FRAM
Sheet: /U_Application MCU - IO/I2C/
File: i2c.kicad_sch
Title: I2C
Size: A4
Date: 2023-04-19
KiCad E.D.A. kicad (6.0.11-0)
Rev: V1B
Id: 5/5
AVED


RESET
USB
POWER
XTAG CONNECTOR
SYSTEM
SYS CLOCK
Sheet: /Application MCU - Misc/
File: Application MCU - Misc.kicad_sch
Title: Application MCU - Misc
Size: A4
Date: 2023-04-19
KiCad E.D.A. kicad (6.0.11-0)
Rev V1B
Id: #/5
AVED

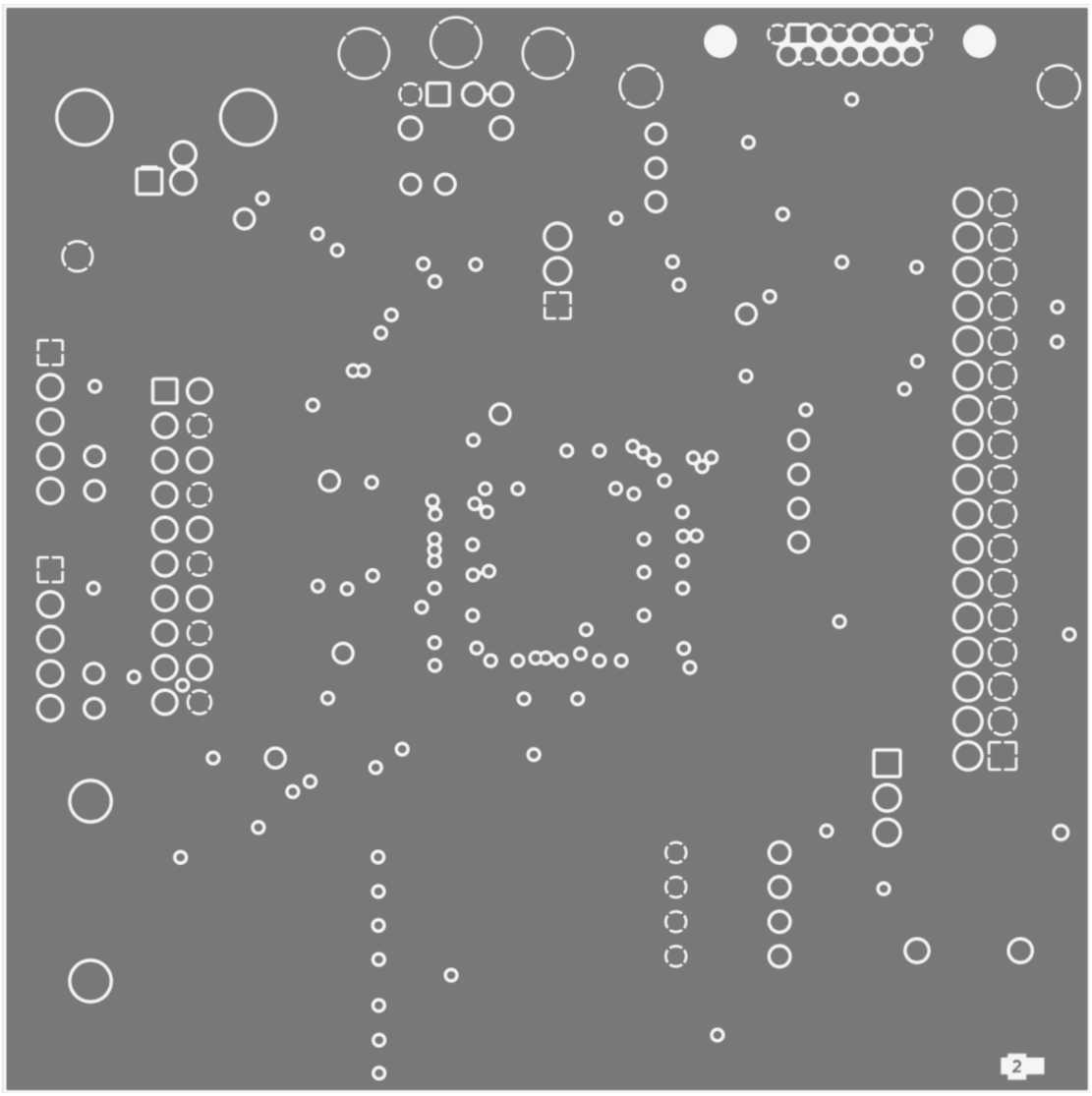
Appendix 8 - PCB plots



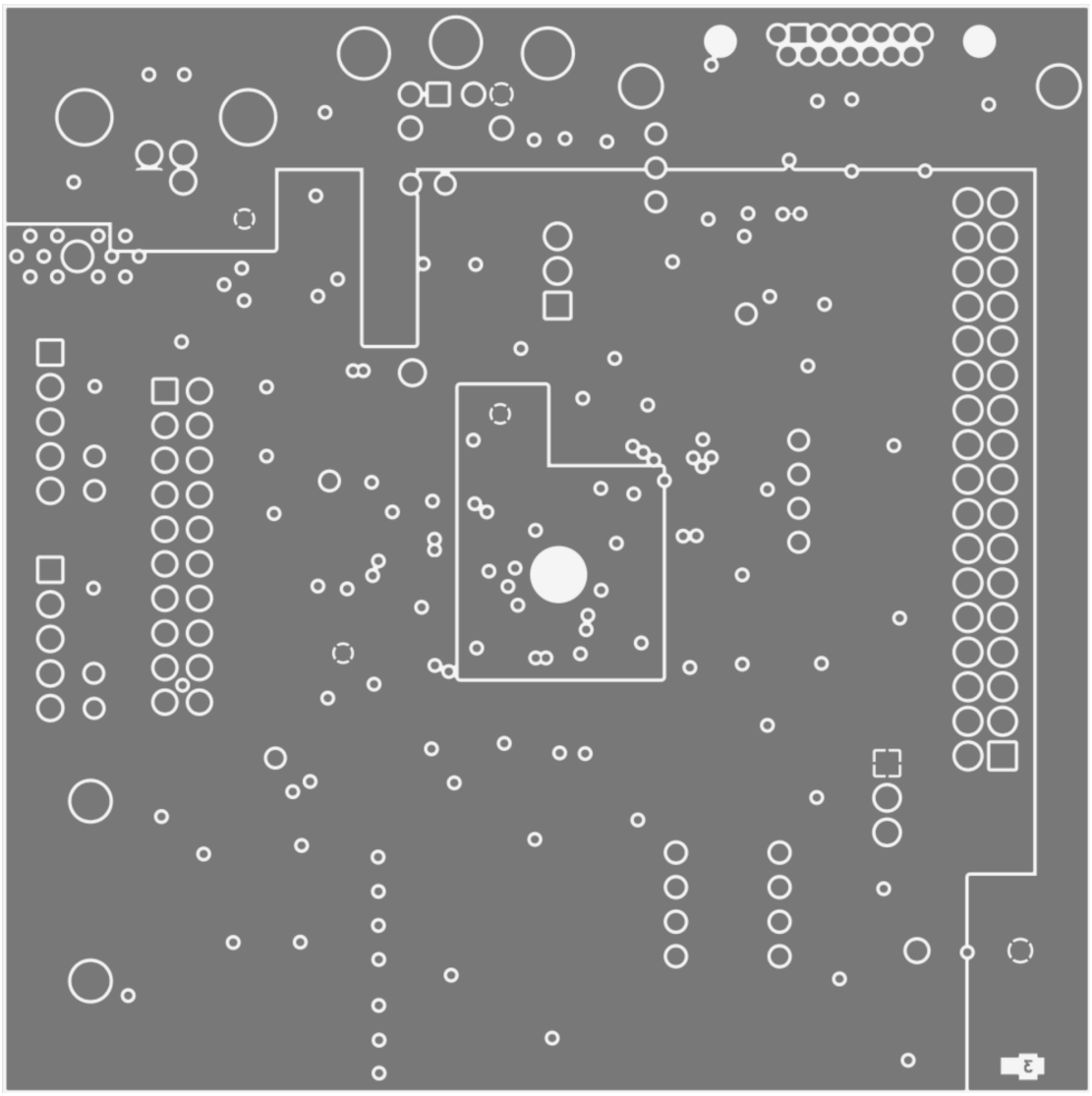
Top view



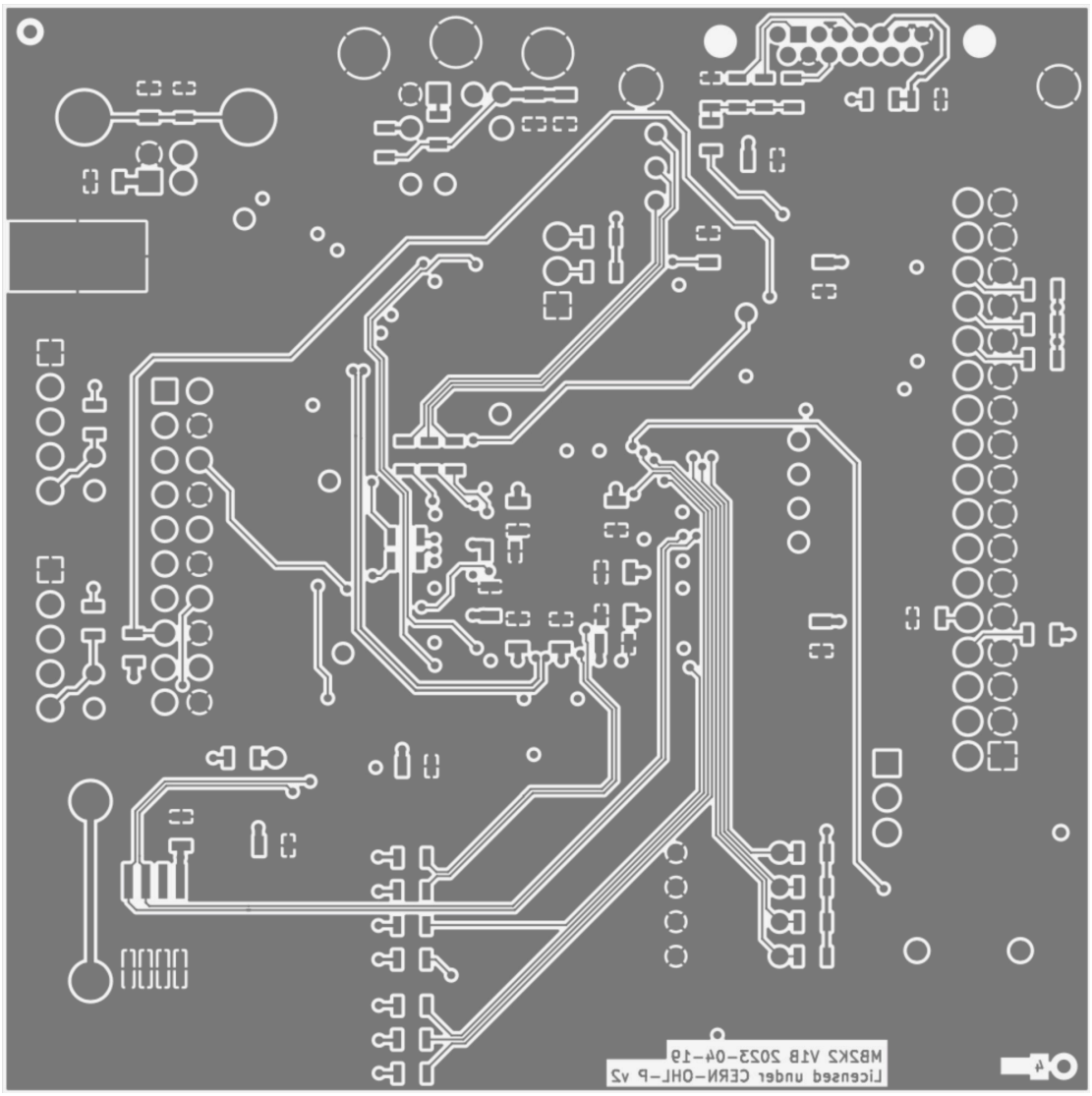
TOP layer



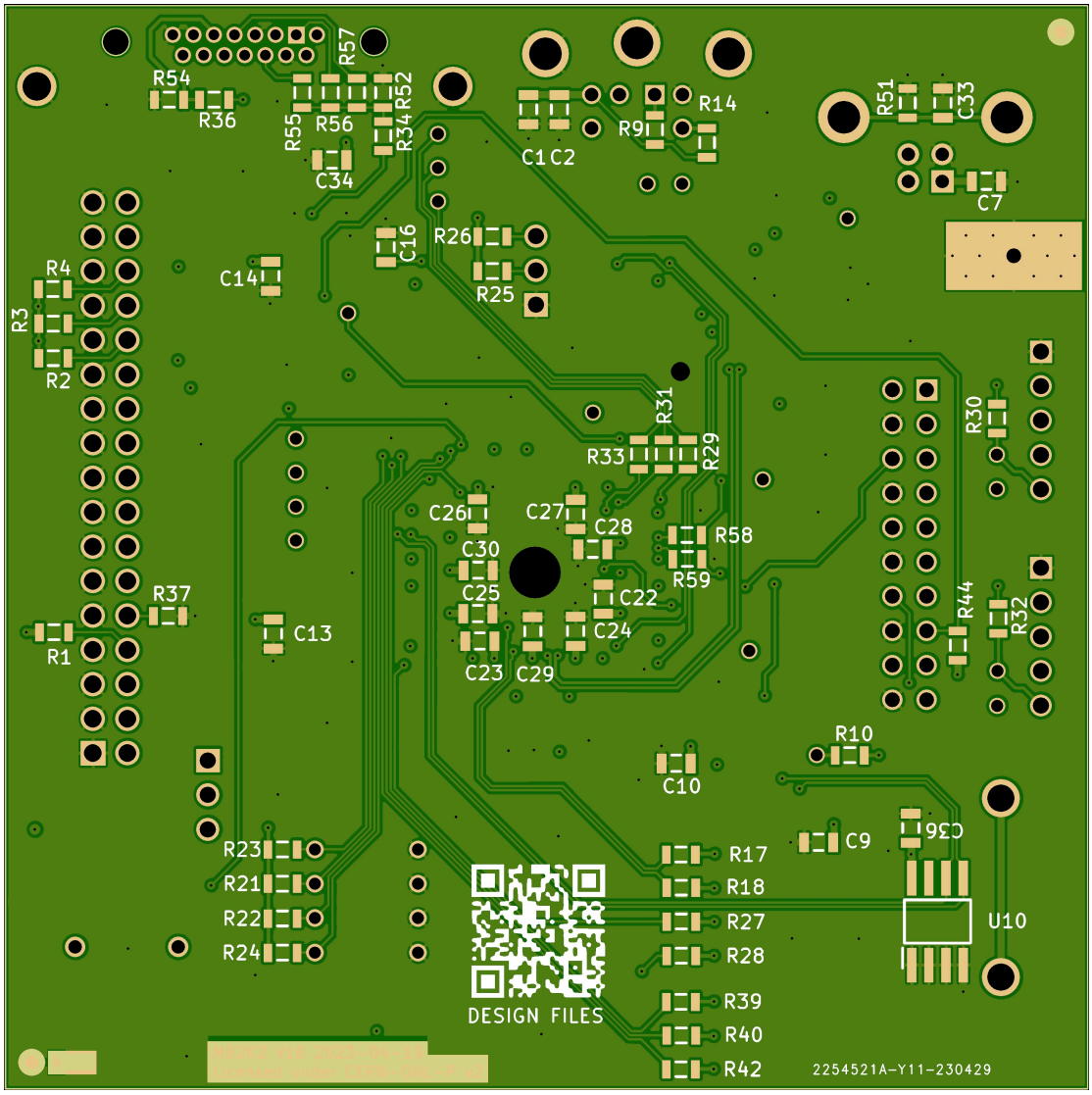
INNER1 layer (Gnd)



INNER2 layer (Power)

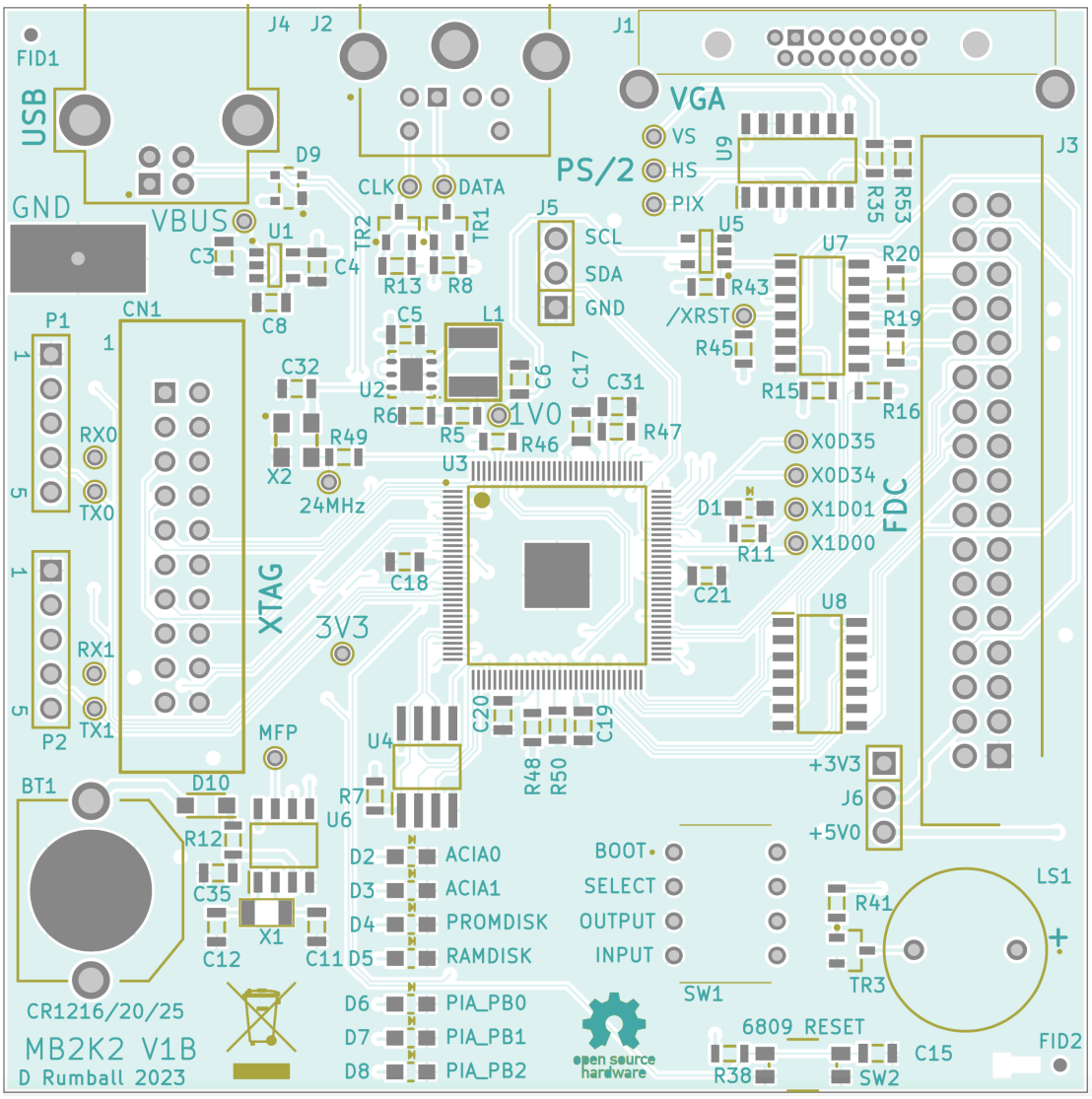


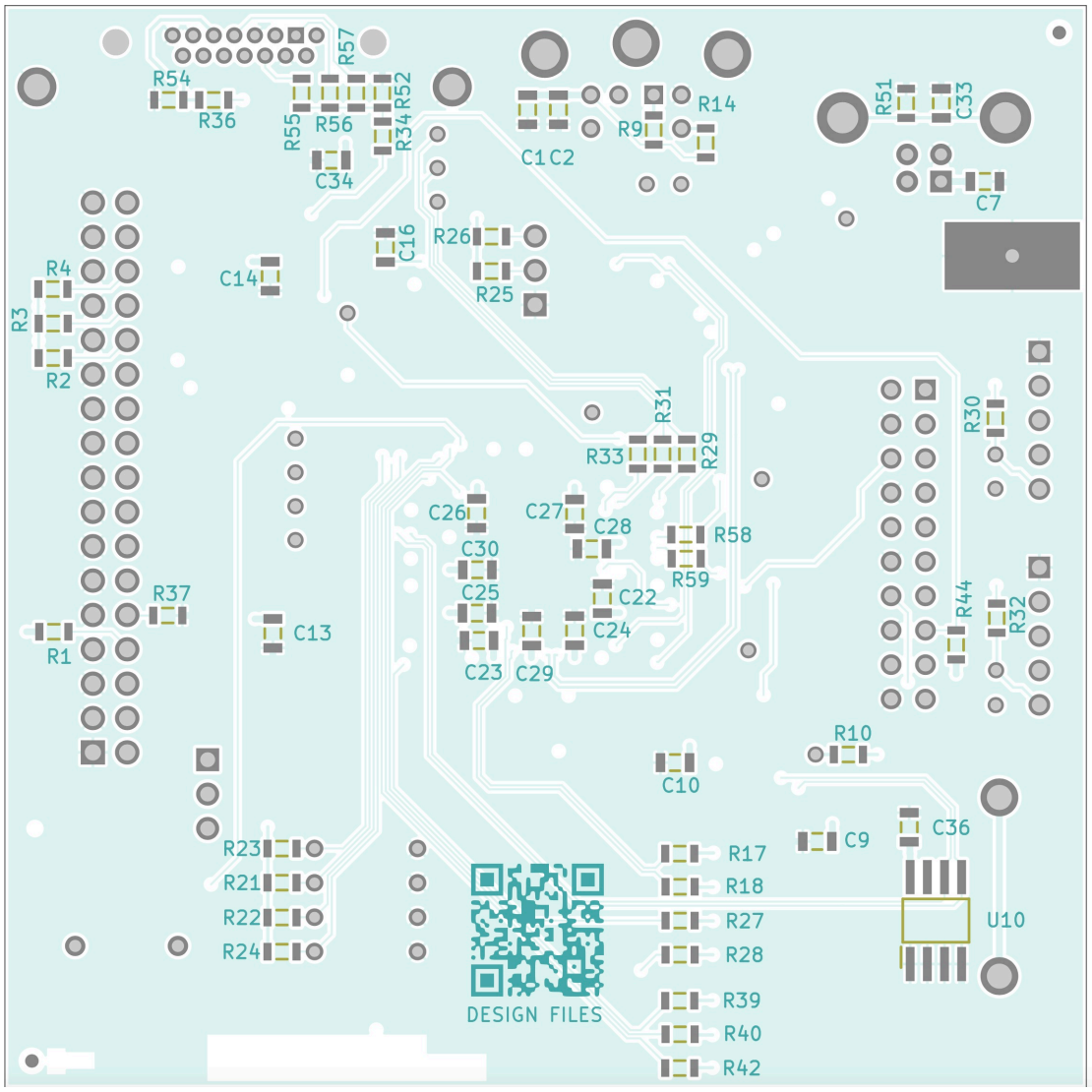
BOT layer


R54
R36
R57
R55
R56
R52
R34
C34
C1
C2
R9
R14
R51
C33
C7
C16
R26
R25
C14
R4
R3
R2
R31
R33
R29
R30
C26
C27
C28
R58
R59
C30
C25
C22
C24
C23
C29
R37
C13
R1
R10
C10
C9
C36
R23
R21
R22
R24
R17
R18
R27
R28
R39
R40
R42
U10
DESIGN FILES
2254521A-Y11-230429

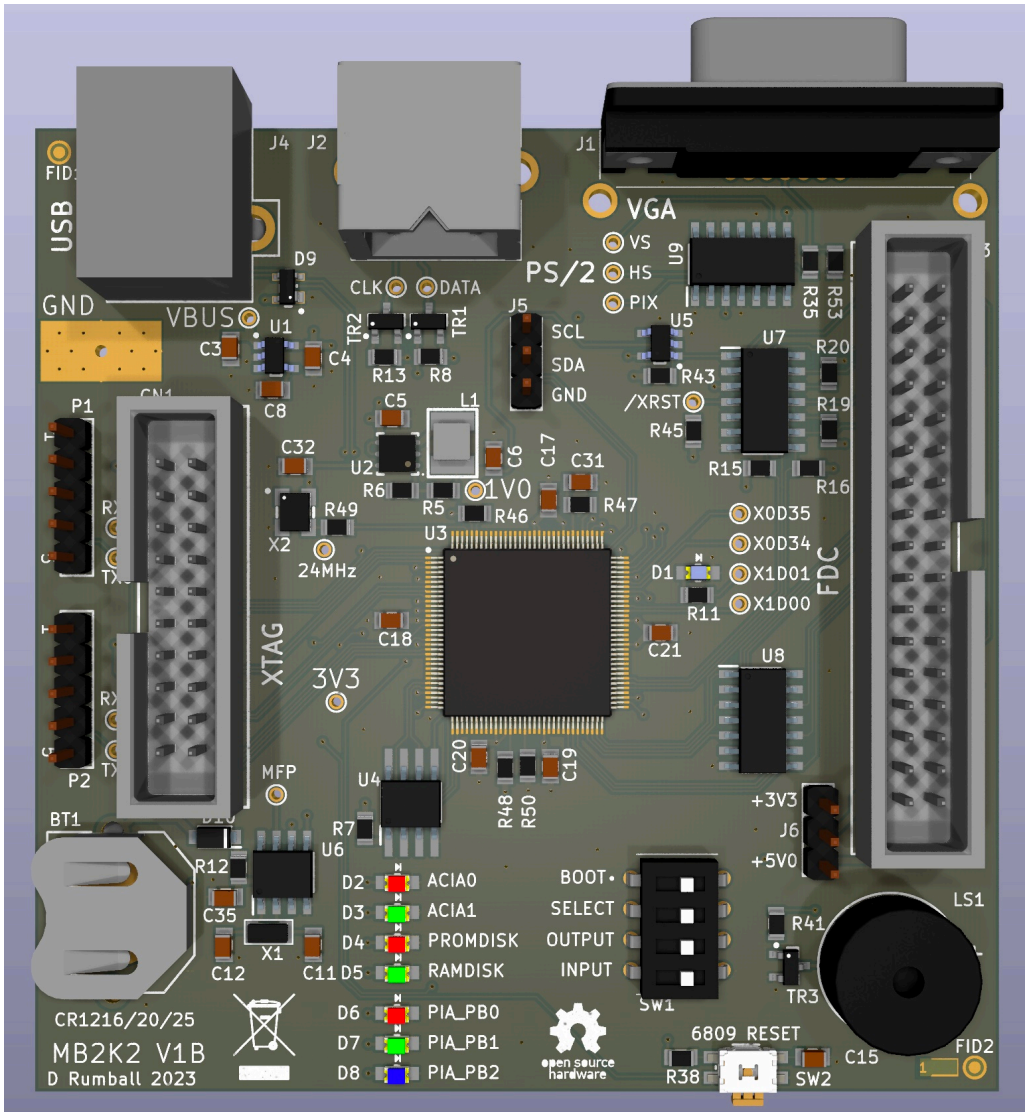
Bottom view

Appendix 9 - Assembly drawings




R54
R36
R57
R55
R56
R52
R34
C34
C1
C2
R9
R14
R51
C33
C7
C16
R26
R25
C14
R4
R3
R2
R31
R33
R29
R30
C26
C27
C28
R58
C30
R59
C25
C22
R37
C13
C24
R1
C23
C29
R44
R32
R10
C10
C9
C36
R23
R21
R22
R24
R17
R18
R27
R28
U10
R39
R40
R42
DESIGN FILES

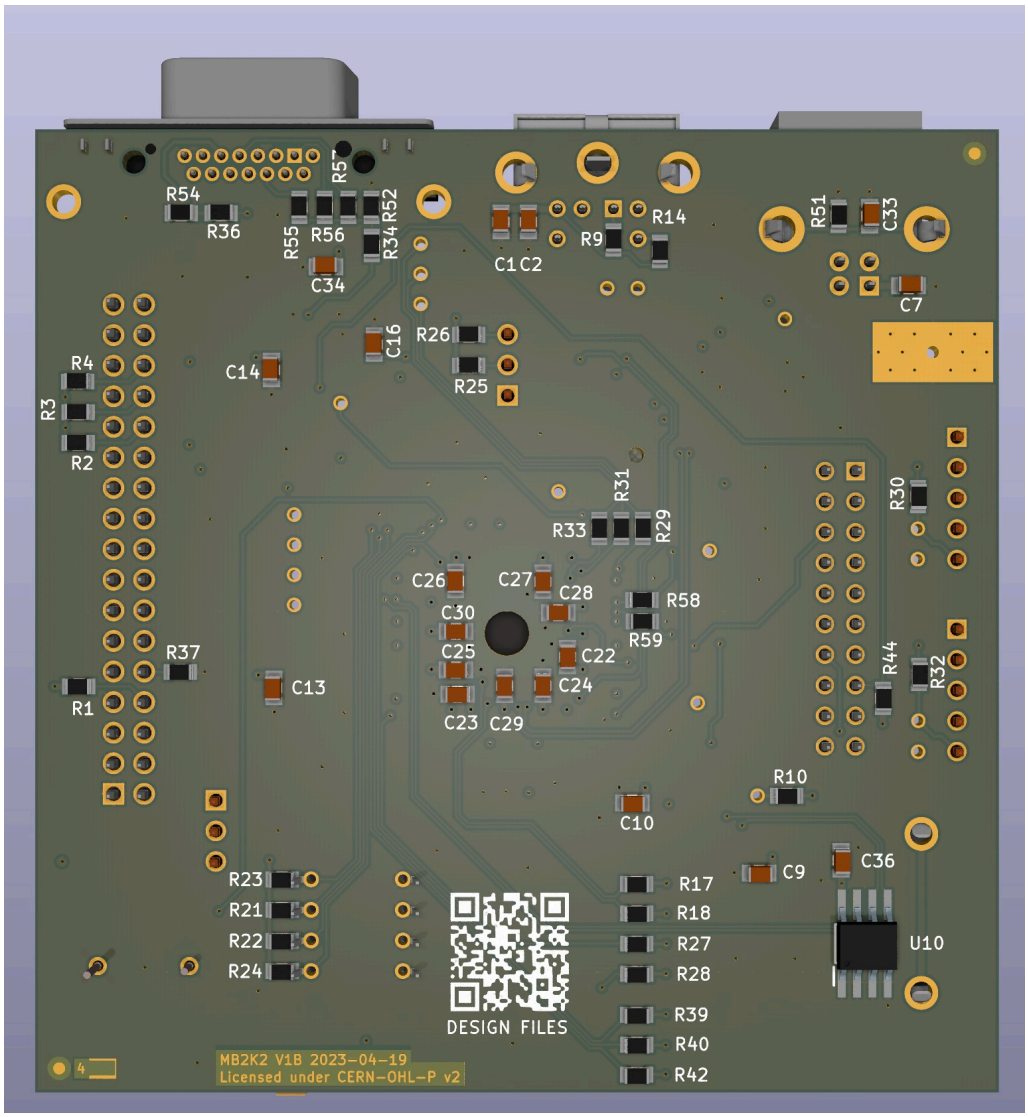
Appendix 10 - 3D renderings



Top view

MB2K2-v1b BOM

Item	Qty	Reference(s)	Value	Footprint	Manufacturer	Manufacturer PN	Description
1	1	BT1	KS3001	CR1216 through hole	Keystone	3001	Battery Holder: THM, 1 Coin Cell, 12mm, PhosBronze/Tin-Nickel
2	3	C1, C3, C5	4u7	C0805	Generic		
3	24	C2, C7, C9, C10, C13, C14, C17, C18, C19, C20, C21, C22, C23, C24, C25, C26, C27, C28, C29, C30, C32, C33, C34, C36	100n	C0805	Generic		
4	1	C4	2u2	C0805	Generic		
5	1	C6	22u	C0805	Generic		
6	1	C8	10n	C0805	Generic		
7	2	C11, C12	4p7	C0805	Generic		
8	2	C15, C16	1n	C0805	Generic		
9	1	C31	1u	C0805	Generic		
10	1	C35	100p	C0805	Generic		
11	1	CN1	70246-2004	20 pin box HDR, 2 row, 2.54mm pitch	Molex	70246-2004	Conn Shrouded Header (4 Sides) HDR 20 POS 2.54mm Solder ST Thru-Hole
12	1	D1	WH	LED0805	Generic		
13	3	D2, D4, D6	R	LED0805	Generic		
14	3	D3, D5, D7	G	LED0805	Generic		
15	1	D8	B	LED0805	Generic		
16	1	D9	PRTR5V0U2X,215	SOT-143B	Nexperia	PRTR5V0U2X,215	Ultra low capacitance double rail-to-rail ESD protection diode
17	1	D10	DSS14U	SOD-123F	Nexperia	PMEG4010EH,115	Diode Schottky 50V, 1A
18	1	J1	634-015-274-992	D-Sub, 15pin, 1.52mm pitch, RA	EDAC	634-015-274-992	D-Subminiature SKT 15 POS 1.52mm Solder RA Thru-Hole 15 Terminal
19	1	J2	5749180-1	Mini DIN Connector, 6pin	TE	5749180-1	DIN Connector, 6 Contact(s), Female, Board Mount, Solder Terminal
20	1	J3	70246-3404	34 pin box HDR, 2 row, 2.54mm pitch	Molex	70246-3404	Conn Shrouded Header (4 Sides) HDR 34 POS 2.54mm Solder ST Top Entry Thru-Hole
21	1	J4	USB Type B	USB Type B	Multicomp	USB-B-S-RA	USB Connector, 4 Contact(s), Female, Right Angle, Solder Terminal, Locking
22	2	J5, J6	61300311121	3 pin HDR, 2.54mm pitch	Wurth Elektronik	61300311121	THT Vertical Pin Header WR-PHD, Pitch 2.54 mm, Single Row, 3 pins
23	1	L1	2u2	Ind 3.8x3.8	Vishay Dale	IFSC1515AHER2R2M01	Inductor Power Shielded Wirewound 2.2uH 20% 100KHz 2.8A 0.045Ohm DCR 1515 T/R
24	1	LS1	3V buzzer	12mm dia, 9mm pitch	CMI	CMI-1295IC-0385T	Audio Buzzer Magnetic 2VDC 5VDC 30mA 3VDC 85dB 2300Hz to 2500Hz Through Hole
25	2	P1, P2	61300511121	5 pin HDR, 2.54mm pitch	Wurth Elektronik	61300511121	THT Vertical Pin Header WR-PHD, Pitch 2.54 mm, Single Row, 5 pins
26	9	R1, R2, R3, R4, R17, R27, R35, R36, R39	330R	R0805	Generic		
27	5	R5, R9, R14, R40, R42	4K7	R0805	Generic		
28	1	R6	17K8	R0805	Generic		
29	1	R7	3K0	R0805	Generic		
30	16	R8, R13, R15, R16, R19, R21, R22, R23, R24, R25, R26, R38, R43, R44, R45, R48	10K	R0805	Generic		
31	2	R10, R41	100K	R0805	Generic		
32	5	R11, R12, R18, R20, R28	1K0	R0805	Generic		
34	6	R29, R31, R33, R49, R58, R59	33R	R0805	Generic		
35	3	R30, R32, R34	100R	R0805	Generic		
36	6	R37, R46, R55, R56, R57	0R	R0805	Generic		
37	1	R47	4R7	R0805	Generic		
38	1	R50	43R	R0805	Generic		
38	1	SW1	210-4MS	8 pin DIP	CTS	210-4MS	Switch DIP ON OFF SPST 4 Raised Slide 0.1A 20VDC 2.54mm Thru-Hole
39	1	SW2	TL1014AF220QG	TL1014AF220QG	E-Switch	TL1014AF220QG	Switch Tactile N.O. SPST Rectangular Button Gull Wing 0.05A 12VDC 1.57N SMD
40	3	TR1, TR2, TR3	TSM2302CX	SOT23	Taiwan Semiconductors	TSM2302CX	Transistor MOSFET N-CH 20V 3.9A 3-Pin SOT-23 Plastic T/R
41	1	U1	SP6205EM5-L-3.3	SOT23_5	Exar	SP6205EM5-L-3-3/TR	SP6205 Series 3.3 V 500 mA SMT Low Noise CMOS LDO Regulator - SOT-23-5
42	1	U2	ST1S09IPUR	DFN6D_N	STMicroelectronics	ST1S09IPUR	Switching Regulator, Current-mode, 1.5A, 1800kHz Switching Freq-Max
43	1	U3	XU216-512-TQ128-C20	TQFP128_XMOS	XMOS	XU216-512-TQ128-C20	XCore XU Microcontroller IC 32-Bit 16-Core 2000MIPS ROMless 128-TQFP (14x14)
44	1	U4	IS25LP032D-JNLE	SOIC-8N	ISSI	IS25LP032D-JNLE	NOR Flash Serial (SPI, Dual SPI, Quad SPI) 2.5V/3V/3.3V 32M-bit 4M x 8 8ns 8-Pin SOIC N
45	1	U5	TLV840MADL08DBVR	SOT23_5	TI	TLV840MADL08DBVR	Low-voltage supervisor with adjustable-reset time delay and manual reset 5-SOT-23 -40 to 125
46	1	U6	MCP79410-I/SN	SOIC-8N	Microchip	MCP79410-I/SN	Real Time Clock Serial 64byte Clock/Calendar/Alarm/Battery Backup Automotive 8-Pin SOIC N
47	2	U7, U8	SN74LVC07AD	SOIC-14N	Texas Instruments	SN74LVC07AD	Non Inverting Buffer, CMOS SOIC-14
48	1	U9	SN74LVC125AD	SOIC-14N	Texas Instruments	SN74LVC125AD	Buffer/Line Driver 4-CH Non-Inverting 3-ST CMOS SOIC-14
49	1	U10	MB85RC1MTPNF-G-JNERE1	SOIC-8	Fujitsu		1Mbit FRAM with I2C serial interface, 1.8V, 3V
50	1	X1	FC-135 32.7680KA-A	XTAL_3215	Epson Toyocom	FC-135 32.7680KA-A	Mini SMD Xtl Khz +/-20PPM, -40~85C 9PF
51	1	X2	ASE-24.000MHZ-LC-T	SMD 3.2mm x 2.5mm	Abracon LLC	ASE-24.000MHZ-LC-T	Oscillator 24MHz ±50ppm 15pF CMOS 55% 2.5V 4-Pin SMD


R54
R36
R57
R55
R56
R34
R52
C34
C1
C2
R9
R14
R51
C33
C7
C16
R26
R25
C14
R4
R3
R2
R31
R33
R29
R30
C26
C27
C28
R58
C30
R59
C25
C22
C24
C23
C29
R37
C13
R1
R44
R32
R10
C10
C9
C36
R23
R21
R22
R24
R17
R18
R27
R28
R39
R40
R42
U10
DESIGN FILES
MB2K2 V1B 2023-04-19
Licensed under CERN-OHL-P v2

Bottom view

Appendix 11 - BOM