MC6809/6502 ATX/6U Computer
aka **COLOSSUS**

Hardware Goal:

A Motorola 6809 processor running on a board which will mount in an ATX case or in a 6U card cage. Board dimensions are 160mm x 233.35mm (6.3" x 9.2").

A Propellor subsystem provides a VGA terminal and PS/2[?] keyboard interface.

Timer, parallel ports, and a separate serial port are included.

The design is based on the original "6x0x" board which operates as an ECB peripheral.

A second CPU socket will be provided for either a 6502 or 6802 CPU chip. This chip may be installed and operated if the 6809 is removed.

Software Goal:

It looks like NitrOS9 level 2 is the most powerful s/w we could bring up on this board; but a paged MMU is required. This software goal is now driving the hardware design effort.

Design Strategy:

Recently the project was divided into smaller, more manageable pieces. A master schematic sheet now indexes close to a dozen independent subsystems. If individual subsystems can be handled by persons with expertise in a particular area, this will save passing around one huge schematic. Integration of an update to one subsystem which does not affect other parts of the system will be as simple as replacing one subsystem schematic with an updated version. Kicad facilitates this with the "schematic hierarchy," and this looks to ease some of the integration headaches.

Technical Stuff

CPU-1: The design is for a 2Mhz MC68B09 chip. Note this is not the 6809E. The -E suffix chip is electrically different.

CPU-2: The design is for a 2Mhz 65C02 chip or a 2Mhz MC68B02. It might be a good idea to allow for 1Mhz versions of these chips, especially the 6802.

MMU: Here is an update on the 10/27 design:

page size: 4K bytes

total memory: 1Mb = 512K SRAM + 512K Flash
There is no option for EPROM or for a 1Mb EPROM chip.
I/O on the 6809 is memory mapped;
Booting takes place by reading the word at address $FFFE in the CPU's 64K address space.

To provide for the above I/O specification, and the boot requirement, the following must happen. A page will be set aside for I/O. This means the I/O space is mapped into one physical page.

Similarly, to allow booting, a special page which overlays RAM will map into ROM space. The boot-up configuration of the machine will be, starting at $0000, 56K of RAM, 4K of I/O space, and 4K of ROM; the MMU is disabled.

There is a hardware efficiency in making these two areas adjacent to one another, so at boot time, with the MMU disabled, the I/O page will be at $yE000 and a ROM page will be accessed at $yF000. 'y' is a hex digit which is jumper selectable. It is the area of RAM which is referenced when the MMU is disabled. The most likely candidates for 'y' are: y = $0, y = $8 or y = $F. Software considerations will probably dictate the best value for 'y'.

MMU mapping divides the 6809 address space into 16 pages; viz., $pxxx, where 'p' is the page number, a hex digit, and 'xxx' is an offset from $000 to $FFF into the page. 3 hex digits means 4K page size. The 64K addressable by the CPU chip is called the “Logical Address” space. When the memory map is enabled, a 16-bit logical address is mapped to a 20-bit (1Mb) physical address. Half of the physical address space is ROM and the other half is RAM. The 'y' jumper mentioned above selects whether RAM occupies the lower half of physical memory and ROM the upper, or vice versa. Software can read the 'y' value from an I/O register, address TBD.

Reading the 'y' register will return the byte $yE, which is the page number of the I/O page. By properly setting up the map, system software can allow user tasks to have access to all I/O devices, or no I/O devices (I/O protection). Likewise, system software can map user pages to ROM. These pages are not writable. There is no provision in the current MMU for interrupt (page fault) on a write-protected page, page-not present, or demand-zero page.

N.B.: The jumper block which locates the I/O page also locates the Boot ROM page. Switching between 6809 boot ROM and 6502 boot ROM is a matter of changing a jumper.

MMU Programming: device addresses and usage
The MMU allows unique memory maps to be defined for up to 64 tasks. Each map is 16 bytes; each byte within the map specifies one of 256 x 4K memory pages.

$EA00	Active Task Register (R/W)– defines which memory map is to be used to translate addresses when the MMU is enabled. On readback, bit 7 reflects the state of the MMU (1=enabled / 0=disabled) and bits 5-0 are the current task map number. Bit 6 always reads back as zero.
$EA10	Task Map Setup (W)-- must be set in order to load or read back a particular memory map. Since the Task Register may currently be mapping addresses, a separate register must be used to read or write a particuler map.
$EA20 - $EA2F	Map (R/W) for the task specified by the Task Map Setup register. A total of 16 logical pages; the contents being the physical pages they reference.
$EA30	MMU enable/disable (W); writing $01 enables the MMU, $00 disables the MMU

	This register is not directly readable. Read bit 7 of the Task Register.
$EA38	Physical I/O page (R); reads jumper setting of the RAM overlay physical page where I/O takes place. Page+1 = physical ROM boot page. Always reads back as binary 'yyyy1110'.

On CPU reset, the MMU is disabled. Response to a hardware or software interrupt is jumper selectable. Interrupt response may be to a) disable the MMU, b) clear the Active Task Register, or c) nothing. See jumper #TBD.

<end>