Processor:

Cosmac CDP1801 with reduced instruction set from standard 1802, possibly TA6889/6890 ?

Memory

1k x 8 RAM expandable to 4k x 8 semi-decoded (?)

Interrupt

1 per frame (e.g. 60Hz) at the beginning of the frame.

Clock Speed

Enough for 8 DMA every 2 frames. (503Khz, see below). Say 2Mhz (arbitrary) and use timers in interrupt routine which we know are 60Hz.

Input pins etc.

1 bit : detect keyboard leader/trailer.

Output pins

1 Bit : Setting 32 or 64 pixels

1 Bit : Setting 32 or 16 rows

1 Bit : Speaker output “low fidelity music”, speaker also responds to key with different tones.

1 Bit : Relay control for cassette tape

DMA

Tape and Keyboard/Punch reader do DMA in depending on the input state (alternate bytes or shift+digit ?). This cannot (obviously) collide with the DMA out for the video display. Therefore the logical thing to do is to delay DMA in until the beam is past the end of the display.

*Alternatively.* This only applies in “Load mode” (DMA in) and the input gate can be read via an Input Port as on Elf machines. This is simpler but requires more hardware (you still need the DMA in whatever).

Display controller does DMA out but does not need to be set up , no timing issues, it buffers the data and displays it itself.

Clock Speed

Enough for 8 DMA every 2 scan lines.

Scan line frequency on NTSC = 60 x 262 = 15,720Hz

2 Scan lines frequency is half this, e.g. 7,860Hz.

8 DMA Outs requiring 8 Cycles each = 503,040Hz.

Controls

Load : reset CPU and accept DMA from cassette, keyboard or punched card reader. Does not function as a “Load switches into memory” element. In this mode (?) the display is likely to be switched off , otherwise DMA caused by the screen display and the input devices will fight ?

Run: reset and run from $0000, and enable display (?)

Keyboard Mode: Specifies whether does double key latch input or shift + hex code input.

16 Hex keys

Shift key

Video Handler

The video circuitry does most of the work, and the processor just has to set R0 accordingly. The function of the interrupt handler would be to load R0 with the display pointer – and nothing else ☺

No other functionality is required providing R0 is not modified, as the DMA in will simply take over when required. This is actually much more efficient than a 1861. As long as R0 points to the data at each new-line-grab-time, it doesn’t care what it is doing when it does the DMA out, it just grabs 4 or 8 bytes from R0 and processes in the background.

Also found comment in another document about dot based system where “the interrupt sets the display pointer”.

Dealing with “unknown information”.

*Assume the block DMA in*. Data is available at the start of the interrupt routine, and is made available via another method. A given register is the current video pointer, this is “assumed” to be correct at the start of interrupt. R0 is automatically set to its end value.

*Routine to play a sound of a given pitch for a given time.*

*Routine to set up the display as either 64x16 64x32 or 32x32 (or 32x16 I suppose)*

**Notes from the document**

*From the timeline this presumably is the 1801 ? Or a TTL based implementation of the architecture. Clearly some issues (e.g. 60/68)*

Low fidelity music synthesis

Audio cassette player that can load programs in 30 seconds. It uses a fault tolerant pulse counting technique.

A relay is provided that permits the cassette output to be connected to a speaker under program control.

The flip flop drives the speaker (is this actually a 1802 and this is the Q line ?)

16 position keyboard (with overlay cards). Each key has an audible click feedback.

The keyboard causes an 8 bit byte to be stored for each depression. The MSB is normally 0000 but can be 0001 in conjunction with the shift. ALTERNATELY. They can be alternate bytes as on the ELF.

*This only works if it post increments, which is part of DMA.*

*How is this disabled when the screen is being displayed*

Load and Run keys . Load Initialises Cassette loading (via DMA ?), Run executes.

*UMpp36 describes such a system. Tape must be doing DMA In ?*

1024 bytes of RAM. There is a design in the UM.

Punched card reader and manual punch which is read vertically.

The card operates through the keypad

Possible expansions : RAM/ROM, Relays

Eliminates the need for ROM

*No ROM*

Program loading and display refresh via DMA

Utility routines and optional hardware are provided

Screen sizes are 32x32 64x16 and 64x32 occupying 128 bytes or 256 bytes

There is a display pointer

*Which must be R0.*

Dot pattern. Two blank rows where DMA reads in “Up to” 8 bytes into a row buffer.

*How does it know whether 4 or 8*

In the next two rows they are displayed as dot patters (see Figure 3)

*Presumably 32 bit doubles the spacing ?*

*During this time the CPU is in Idle mode, e.g. it does 4 to 8 DMA reads then IDL again ? so a display routine is a series of IDLs.*

*How does it know whether 32 or 16 pixels vertically ?*

Each TV line is 65us – the data is read “during this time”

There is an INT at the “beginning” of each TV frame.

*Does it actually mean this, or is this like the interrupt in the 1861 ?*