Understanding the Amstrad CPC Video, RAM and Gate Array Subsystem

Mike Sutton ⋮

The Amstrad CPC is my favourite[1] 8-bit home computer system – I still have my childhood CPC464. There are a number of things I love about it's design, including the video system, which offers higher resolution and a larger colour palette than most of it's siblings.[2]

Recently I've put some effort into understanding how the CPC system works, and in particular the video and RAM circuitry. As part of a project to emulate the gate array with a Raspberry Pi Pico/RP2040 I've taken an especially deep dive into this area. This article is my current understanding of how the system works.

Before proceeding I want to give credit to a small group of people who have de-capped and reverse engineered an actual gate array. Reading their schematics have given me a tonne of insights into how this all fits together and saved me a lot of reverse engineering work.

RAM-Video System Overview

Let's begin by taking a look at what the RAM-video[3] system has to do.

The Z80 processor uses the system's RAM to store data and some of that data is the video data which is displayed on screen. The CPC uses a 6845 CRTC controller to assist with converting that data into a video signal, along with custom circuitry contained in the 'gate array',[4] or GA, chip.

This means that both the Z80 and the 6845/GA need to access the RAM.[5] The system designers need to find some way to ensure that the RAM can be shared between both systems without issue. For example, as we shall see, the video system needs to access RAM at high speed and with precise timings to generate a high quality image. And the Z80 needs to be able to read from and write to RAM at specific points on it's own operation cycles.

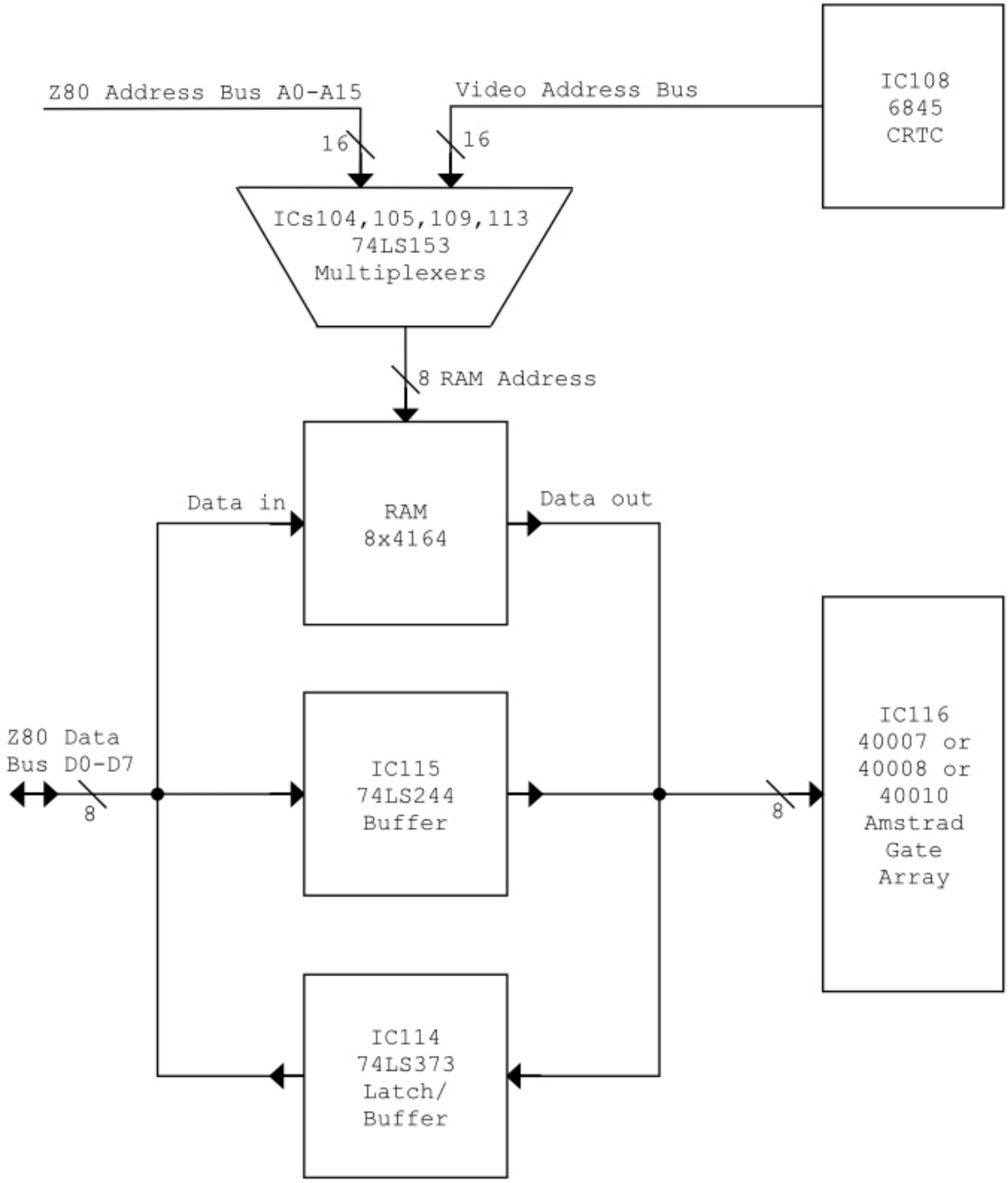

Fig 1. Block diagram of the Amstrad CPC RAM-video system

Figure 1 shows a block diagram of the RAM-video system (see also the schematic). At the top is the 16-bit Z80 address bus and the 16-bit video address bus coming from the CRTC. A multiplexer switches between these two sources to generate an address which is sent to the RAM chips. However, the RAM chips actually only have an 8-bit address bus. 16-bit addresses need to be passed in in two halves (I'll describe later exactly why). So the multiplexer actually divides between the upper and lower halves of the address bus and the upper and lower halves of the video address bus. Thus giving it four possible outputs: video address high; video address low; bus address high and bus address low.

If the address passed into the RAM is a video address (from the CRTC) then the output from the RAM chips will be read by the gate array (and turned into a video signal). If the address is from the z80 then it could either be a memory read or a memory write. The memory chips used have separate pins for data in and data out. For a memory write the

data will be read in directly from the Z80 data bus. For a memory read it will be latched into the 74LS373 (IC114) where it will wait to be read by the Z80. Finally, the Z80 also needs to be able to send configuration data (such as the current screen colours) to the gate array. This is done via the 74LS244 (IC115) buffer. When this device is enabled signals from the Z80 data bus pass through it and can be read by the gate array.[6]

Video Timings

The CPC has three video modes:
Mode 0: 160w x 200h @ 4-bits per pixel (sixteen colours)
Mode 1: 320w x 200h @ 2-bits per pixel (four colours)
Mode 2: 640w x 200h @ 1-bit per pixel (two colours)

Those bit depths mean that each byte of video memory encodes two pixels in mode 0, 4 pixels in mode 1 and 8 pixels in mode 2.

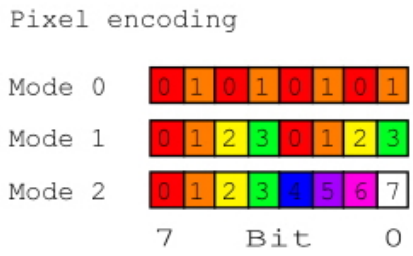

Fig 2. How pixels are encoded into a byte of screen RAM. The numbers are the index of the pixel within the byte, where the 0th pixel is the first to be displayed, and the left-most pixel on screen.

If we divide the pixel widths of each mode by the number of pixels per byte we get:
Mode 0 = 160 / 2 = 80 bytes
Mode 1 = 320 / 4 = 80 bytes
Mode 2 = 640 / 8 = 80 bytes

Which is interesting. Every screen line requires the same number of bytes of storage. A naive implementation of the gate array might read in a byte of data for each and every pixel. But it's far more efficient to read in a byte once and extract data for each individual pixel out of it. Thus the gate array always reads in bytes at the same frequency no matter what the video mode.[7] It then varies the length of time that each pixel is displayed on screen. Thus in mode 1 it displays each pixel for twice as long as it does in mode 2, and in mode 0 it displays each pixel for four times as long.

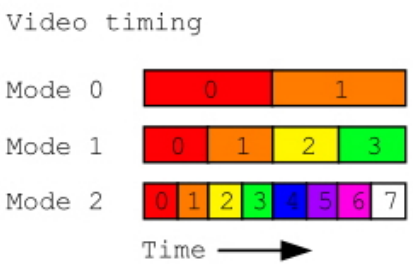

Fig 3. The sequence and length of time each pixel is displayed in each video mode. This also corresponds to the width of each pixel on screen.

The video system outputs a 50Hz signal suitable for PAL or SECAM TV systems.[8] I won't do the full maths but with 640 screen pixels[9] per line, plus border areas and 'blanking intervals' between lines and screens the gate array needs to output a new screen pixel every 16-millionth of a second. Or, in other words, it has a 'pixel frequency' of 16 MHz.[10]

And since each byte of video data encodes eight screen pixels (taking screen mode and the stretching described above into account) we can divide that 16MHz by eight to get the rate at which data needs to be read from memory: 2MHz. Phrasing that differently gate array needs to read in a new byte of video data every 2-millionths of a second.

With the gate array reading data at 2MHz and the CPU running at 4MHz we can see that there is a very real need to manage access to the RAM to avoid clashes.

RAM and RAS and CAS

There's one more thing you need to understand before we can delve into the full details of the control signals: early RAM chips, such as those used in the CPC and other computers of the era, only had eight address lines.[11] The Amstrad used 4164 ICs. These are 64k x1 chips. Which is to say that each chip stores 65536 *bits* of data with a 1-bit wide data bus.[12]

65536 equates to a 16-bit wide address. The chips of the time used what is called 'multiplexing' of the address lines: they latched in 8-bits, then read in the other 8-bits, and then output the data bit at that address (or wrote data to that address). These are referred to as row addresses and column addresses. You can think if them like the rows and columns of a display: select the row to access from and then select the column within that row.[13]

In hardware terms, the chips had a /RAS input line and a /CAS input line. Take the /RAS line low to latch the row address and the /CAS line low to read or write the data.

The 4164 chips have one other feature, called 'Page Mode'. This allows faster access to data within a single row. Again you assert the /RAS signal to latch a row address and then assert /CAS to read or write a bit. You can then release /CAS, change the address lines and assert /CAS again to access the another bit of data without having to re-latch the row address. This is a convenient way to save time when accessing multiple bits of data within a row, and we'll see later how it gets used in the CPC.

The Gate Array Signals

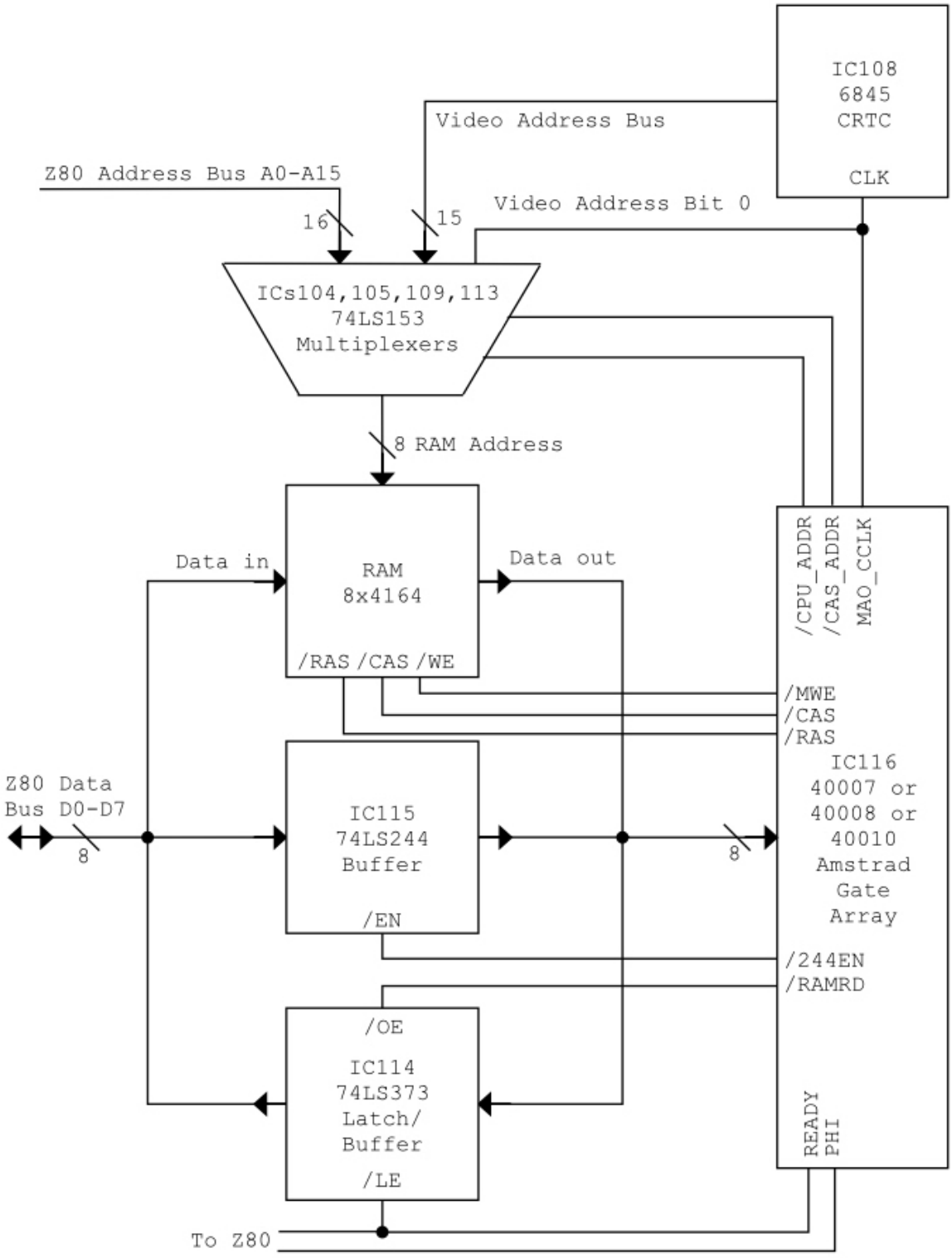

Fig 4. Block diagram of the CPC's RAM-Video system showing the control signals output from the gate array.

We know know enough about the system to start describing the control signals. Figure 4 above shows the same as Fig 1 but with the addition of the relevant control signals output by the gate array. For the gate array I've only shown the inputs and outputs which are relevant to the RAM-video system.

Starting with the multiplexer at the top. /CPU_ADDR and /CAS_ADDR select which of the four address sub-sets will be output to the RAM. If /CPU_ADDR is low then data from the z80 bus will be output, otherwise signals from the

CRTC. /CAS_ADDR switches between a row (upper 8-address bits) and column (lower 8 address bits) as suitable for the /RAS and /CAS inputs to the RAM.[14]

In practice on fifteen of the video address lines come from the CRTC. /CCLK provides the sixteenth address line – the least significant bit. Working as video address bit zero this line selects between two neighbouring RAM addresses. Remember the 'page mode' of the RAM chips? The gate array clears this bit and sends the /RAS and /CAS data to the RAM to read a byte. It then sets this bit and sends another /CAS pulse to the RAM to read the next screen byte. I'm not totally clear why they chose to do this rather than sending the bit from the CRTC and clocking the CRTC faster. Possibly the CRTC can't be clocked fast enough.[15] Possibly it made the gate array simpler. Certainly it guaranteed that both bytes would be on the same row address and enabled the use of the RAM's page mode to reduce access times.

/CCLK also provides the 1MHz clock signal to drive the CRTC.

The RAM has /RAS and /CAS inputs as described above. The /MWE signal drives the /WE (write enable) pin low when the z80 is writing data to RAM.

The /244EN signal activates the 74LS244 so the gate array can read the Z80 data bus.

When the processor wants to read from RAM the data is read via 74LS373 (IC114). When the /LE pin transitions low the chip latches the data at it's input pins. This data is output when the /OE pin is low. If /OE is low whilst /LE is high then data passes through. The Amstrad uses the READY signal from the gate array to drive the /LE function and the /RAMRD signal to drive the /OE pin. Thus, data is latched when the READY signal transitions low. The /RAMRD signal can occur at any time and simply depends on the Z80's /RD and /MREQ signals, which ROMs are currently enabled and the actual address.[16] As it's timing isn't affect by the GA and the other signals I won't be describing it any further here.

The system also sends the PHI and READY signals to the Z80 and the expansion bus. PHI is the 4MHz clock to drive the CPU. READY is connected to the /WAIT signal of the CPU and I'll be describing it in more detail later.

There are a number of gate array inputs which are of relevance (not shown on the diagram for clarity). XTAL is the 16MHz clock signal from the crystal circuit. The other signals come from the Z80. /MREQ is active when it wants to access memory (reading or writing). /IORQ is active when it wants to access I/O. /RD is active when it wants to read from either memory or I/O.[/efn_note]There is an edge case when this isn't true but that isn't relevant to the gate array's operation which we are discussing here so I shall ignore it.[/efn_note] (And by implication, if /MREQ or /IORQ is active and /RD is not active then the CPU is writing to memory or I/O. This fact is used by the gate array to save it needing an extra pin for the z80's /WR signal).

READY and the Z80's /WAIT Input

At this point we can finally turn our attention to the method used by the gate array ensure that the CPU and video system are not both accessing memory at the same time. The CPC's designers did this by occasionally 'pausing' the Z80.

Older memory wasn't especially fast. And sometimes I/O devices can be too slow for a CPU running at multi-megahertz speeds, so the Z80 has an inbuilt 'pause' mechanism – the /WAIT input. If /WAIT is asserted at the appropriate moment the Z80 will literally just stop whatever it's doing for a clock cycle (or until the /WAIT signal is released). I won't describe the mechanism in detail here but you can see how it affects the timings here.

Every Z80 operation takes multiple clock cycles to operate. It divides each operation into one or more blocks which are referred to as 'M states'. Each M state includes a single input or output operation or memory read or write,[17] and each is either three or four clock cycles long. The CPC generates the /WAIT signal in such a way that three clock cycle M states get stretched to four clock cycles whilst the four clock cycle ones are unaffected. To do this the GA generates the READY signal, which is connected to the Z80's /WAIT input.[18]

Timing Diagrams

Now let's take a look at some oscilloscope traces which show how this all works in practice. The 'scope traces show the 16MHz crystal (XTAL)[19] and 4MHz PHI.[20] CPU clock signals at the bottom. Above that are the /CPU_ADDR, /CAS_ADDR and /CCLK signals controlling the multiplexers. Then the /RAS and /CAS signals to RAM. Above those are the /244EN, /MWE, /RAMRD and READY signals which control the chips around the gate array. Finally, at the top, are the signals coming into the gate array from the Z80: /RD, /MREQ and /IORQ.

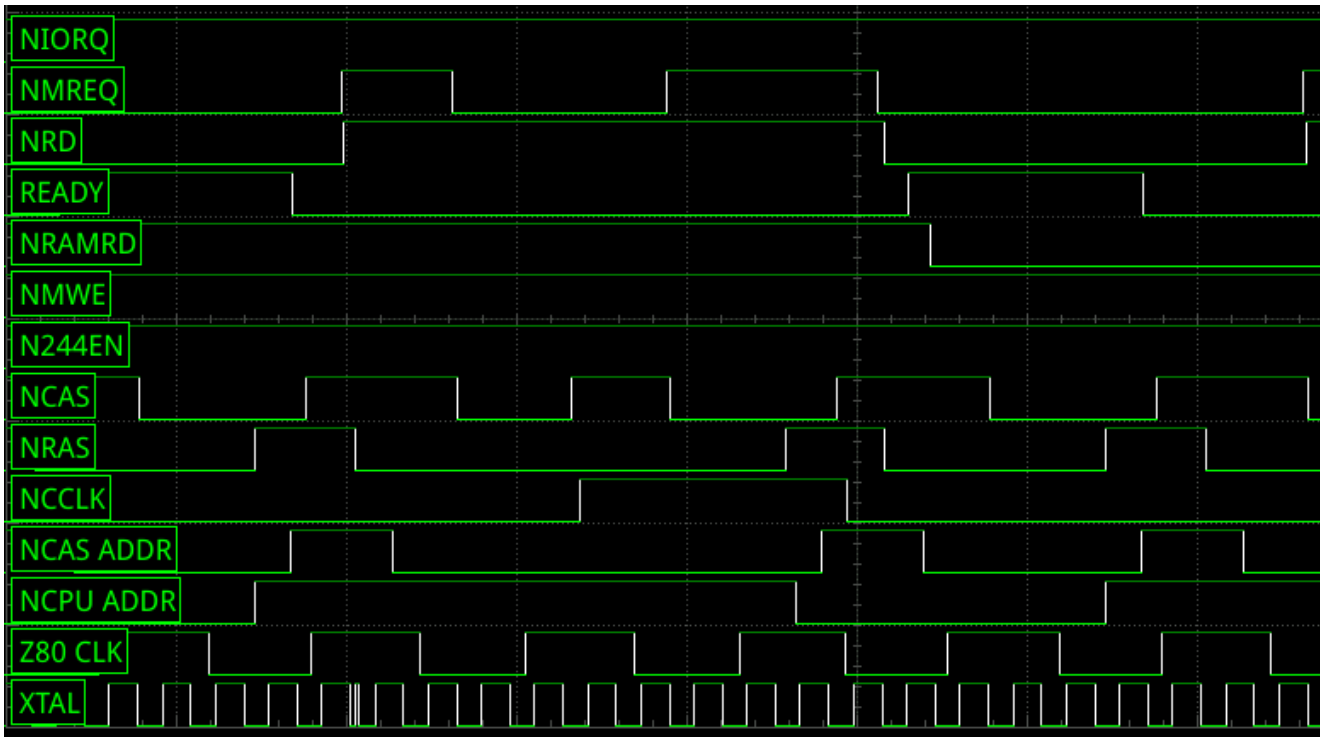

Fig 5. Timing sequence while reading from RAM (/MREQ and /RD go low, as does /RAMRD)

Figure 5 shows an oscilloscope trace of a typical operating sequence – in this case for the processor reading from RAM. The gate array operates on a sequence which takes place over 16 cycles of the XTAL 16MHz clock.

The sequence starts at the point that /CPU_ADDR goes high with /CAS_ADDR also going high shortly after. With both signals high the RAM is being fed the video row address. /RAS is initially high and drops low to tell the RAM to latch in that row address. /CAS_ADDR transitions low soon after and the RAM is fed the column address. Note that /CCLK is low, so video address bit zero is low and the RAM is being fed the address of the first of the two video bytes to be read. /CAS then drops low and over the next clock tick or so the RAM will be outputting the video data and the gate array will be latching it in.[21]

The gate array now prepares for the second byte of video data. /CCLK goes high as does /CAS in readiness to use the RAMs page mode feature. /CAS goes low again and the second byte is output by the RAM and latched in by the gate array.

At this point the CPU is now free to access memory and /CPU_ADDR transitions low so that memory will be fed an address from the Z80 address bus. There's actually four things that could happen now:
1) The CPU could write to RAM.
2) The CPU could read from RAM.
3) The CPU could read to or write from an I/O device, including the gate array.
4) None of the above. (This could happen if the Z80 doesn't make an I/O or memory access on that cycle or if some external hardware has paused the CPU).

If the CPU is reading from RAM (as shown in the scope trace above) then we'll see a repeat of the /RAS high => /CAS_ADDR (and /CAS) high => /RAS low => /CAS_ADDR low => /CAS low sequence to get RAM to output the requested data. By the time /CAS transitions low /RAMRD will be low and READY will be high and the data ouput

from the RAM will be passed on to the Z80 data bus. When READY transitions low the data will be latched into the ‘373 so the gate array can advance to the next cycle. /RAMRD will remain low for a while (until the Z80 cancels the /RD and /MREQ signals) so the Z80 has time to read the data.[22]

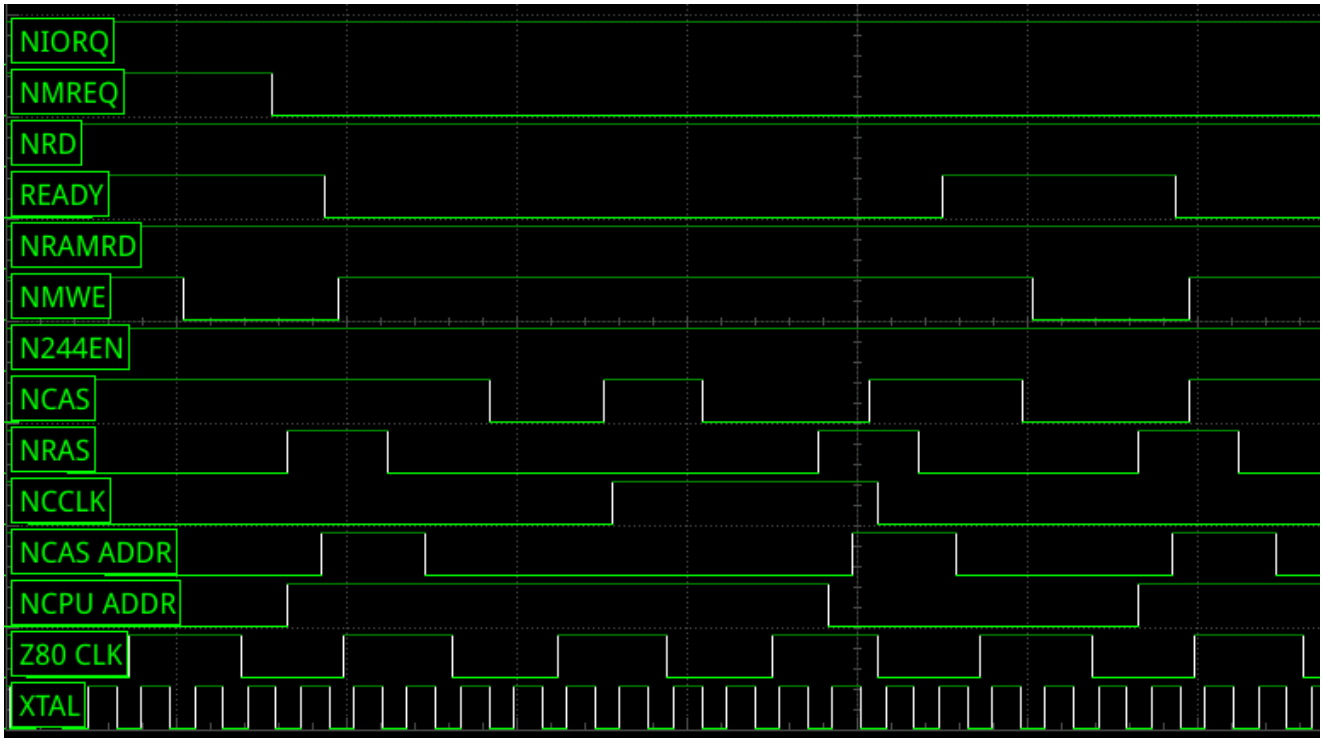

Fig 6. Timing whilst writing to RAM (/MREQ low and /RD high, /MWE goes low)

The sequence for writing to RAM is very similar but in this case /RAMRD stays high and /MWE will go low so the RAM will latch data into the given address (remember that the data in pins of RAM are connected directly to the Z80 bus). Note that the /MREQ signal is low for much longer than that shown on the memory write cycle shown above. This could be because the Z80 uses a shorter read time for an M1 cycle (essentially when reading an opcode) than other memory reads. The longer read could also be because the cycle is being stretched by the /WAIT (READY) signal.

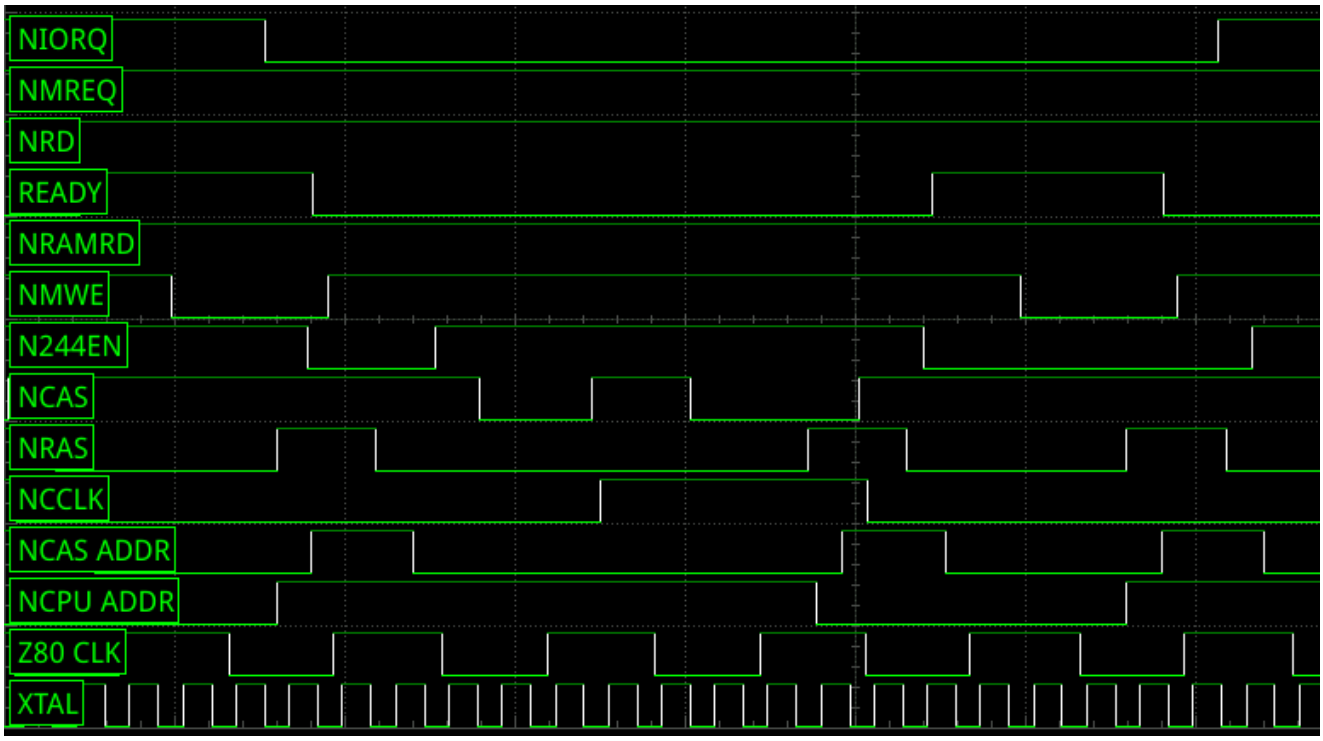

Fig 7. Timing during an I/O write (/IORQ low and /RD high, /244EN goes low even if not writing to the gate array)

For an I/O write we'll still see the second /CAS cycle does not happen. Instead the /244EN signal goes low and the '244 outputs data from the z80 address bus which can be read by the gate array. Interestingly the trace shows that the second /RAS pulse occurs as does the /MWE pulse. This is, presumably, to keep the gate array logic simpler, and it doesn't affect the RAM which doesn't output or write unless the /CAS signal activates.

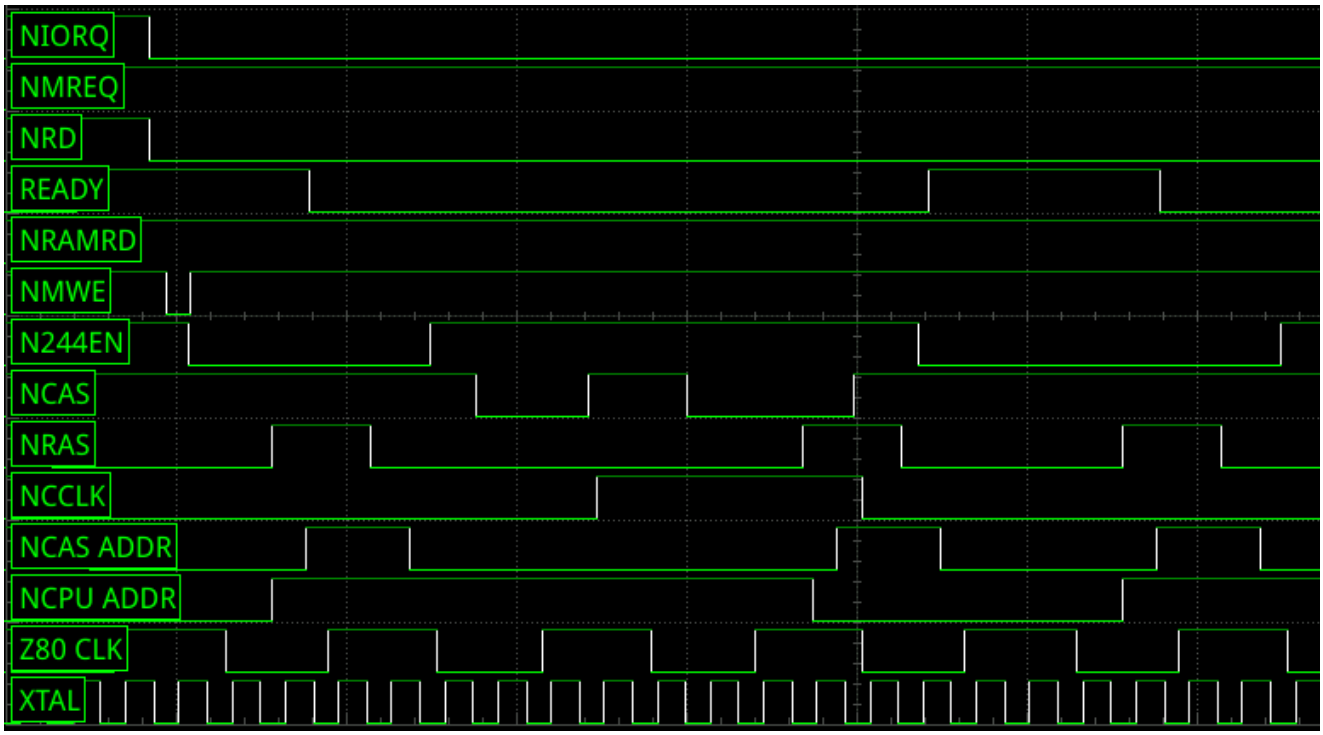

Fig 8. Timing during an IO read (both /IORQ and /RD low. /244EN goes low but has no effect)

And, similarly, on an I/O read the /244EN signal still activates because it keeps the logic simpler and doesn't affect anything.

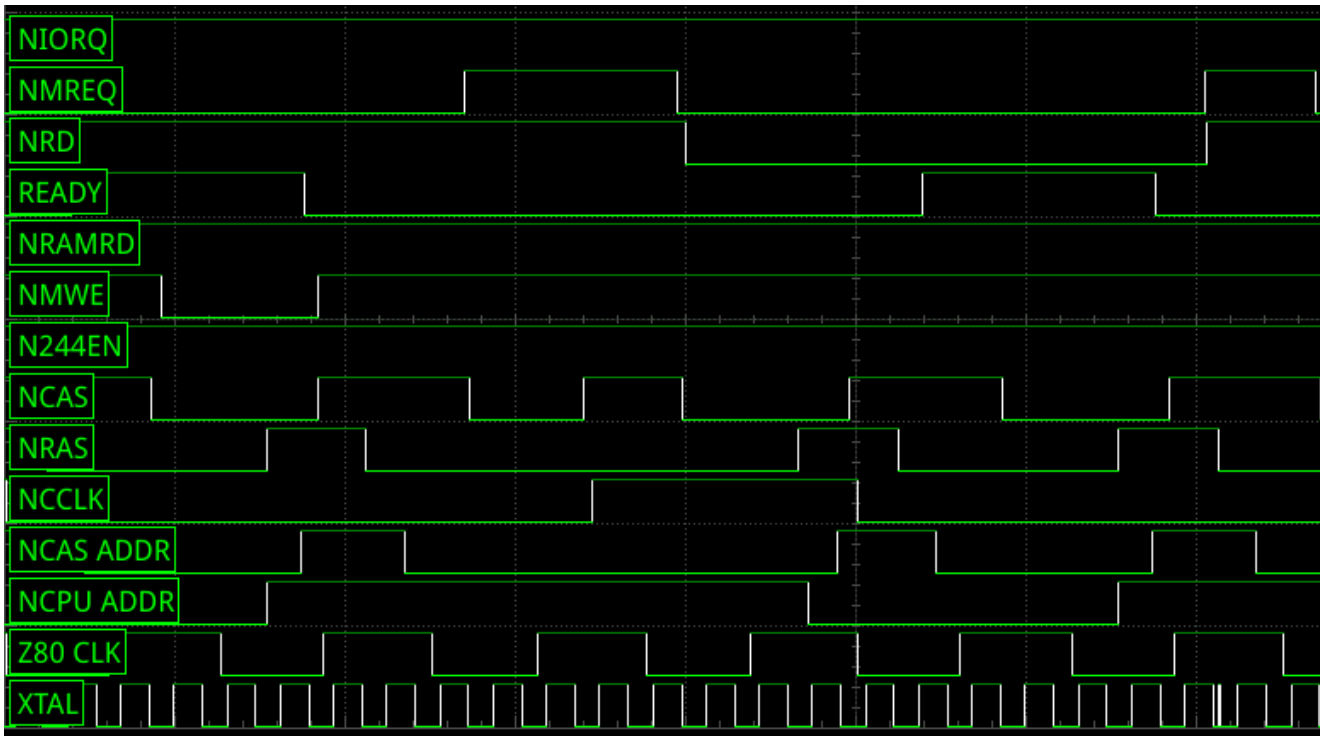

Fig 9. Timing during a ROM read (/MREQ and /RD go low but /RAMRD stays high)

Here's a trace which shows the Z80 /RD and /MREQ signals activating but the gate array doesn't activate /RAMRD signal. This indicates a read from ROM.[23]

If no I/O or memory access is happening then, as might be expected, the CPU's CAS cycle doesn't happen and /MWE and /244EN both stay high.

Returning to the general operation of the gate array, the final step is for /CCLK to return low ready for the next video read. Since /CCLK is also the clock for the CRTC this also causes it to advance to the next step in it's own operations (usually the next memory address to be read).

Summing Up

I think it's clear that designing a production home computer is a very technical task requiring a lot of skill. But it's not until you take a detailed look at a real-world design such as this that you realise just how much skill is involved. At first glance the CPC's gate array appears to simply be marshalling the video data into pixels on the screen with a couple of side tasks thrown in for good measure.

But it also needs to do the complex tasks of managing the RAM access and the associated components of the system. And that requires detailed knowledge of every chip involved, from the Z80 down to the 'glue logic'. And that includes detailed knowledge of the timings required for each device. I've taken a cursory look at some of those timings and there's very little room to spare across the timing sequence.

I've always been in awe of the people who could design something such as this, and looking at the design in detail I'm even more amazed of the brilliance of the computer designers of the era.

Footnotes

1. As a teenager I nearly chose the Elan Enterprise instead of the Amstrad. Having recently bought an Enterprise, that machine may well be my new favourite.
2. Admittedly it was a late-comer on the scene, which meant that it's designers had the chance to learn from the machines that came before, as well as new developments such cheaper RAM prices.
3. Henceforth I shall refer to this circuitry as the 'RAM-video' system for convenience.
4. Officially this is the 'video gate array', or VGA, chip. But since that term is likely to cause confusion with the IBM PC's VGA display standard I shall use the terms 'gate array' or GA.
5. When two subsystems both need to access a resource, possibly at the same time, it is referred to as 'bus contention'.
6. There's no functionality within the gate array for data to be read back by the CPU, so there's no data path or timings available for this. Obviously this could be done via the '373, as with reading from RAM, if it were needed.
7. And this makes the entire video system much simpler than one which processes data at multiple frequencies but with the penalty of a lack of flexibility in the video modes available.
8. The system can also be configured for a 60Hz signal suitable for NTSC systems but I'm not sure if any systems were manufactured with this as the default. If so they would have had the same resolution and pixel frequency but different border sizes to account for the different number of rows and column on the image.
9. I'm talking 'screen pixels' here, rather than the 'logical pixels' of the display mode.
10. If you've ever noticed that the CPC has a 16MHz crystal even though the processor is only clocked at 4MHz, this is the reason.
11. I don't know the exact reason for this but Intel was a major player in RAM chips in their early days and it had something of an obsession with using small chip packages with less pins. Indeed one of the reasons why the first microprocessors, Intel's 4004 and 8008 designs, were difficult to use in systems was because they had very few pins and needed to multiplex the pins (using the same pins for multiple functions)
12. Multiple chips were used in parallel, such as the eight chips used in the CPC464 due to it's 8-bit data bus. (The CPC6128 used 16 chips but this was divided into two banks of eight with only one bank being used at a time).
13. Mutliplexing the address bus in this way also made memory access significantly slower. Although at the time this was probably not a major issue. Indeed the need to multiplex on the CPC didn't significantly reduce the system speed below the CPU's rated 4MHZ.
14. /CPU_ADDR also provides the 1MHz clock signal to the AY-3-8912 sound generator, not shown in the diagram.

15. With reference to the CPC's signal timings (see below), the use of the RAM's page mode means the second byte is being accesses very soon after the first which is probably too fast for the CRTCs 3MHz maximum clock speed. And the multiplexer's are also relatively slow which would not be helpful in getting the second address to the RAM chips in time.
16. It can also be inhibited by the /RAMDIS signal from the expansion bus, which isn't shown in the diagram above for clarity.
17. Although a few M states are spend purely on internal processes and don't have any external activity.
18. As a point of interest the ZX Spectrum also usually resolves bus contention with wait states, but sometimes does it by pausing the CPU clock. (See the book "The ZX Spectrum ULA" by Chris Smith for more details). The BBC Micro (and, I understand the Commodore 64) takes advantage of the 6502's slower clock speed to read video memory during part of a clock cycle, without needing to slow the CPU. Other micros of the era will use similar mechanisms.
19. The traces show some glitches on this signal. I believe that's a sampling issue with the scope, especially as it isn't affecting the gate array's behaviour
20. The gate array actually generates this signal inverted. The trace shows the signal that gets to the Z80 (and the expansion bus)
21. The 4164 RAM chips take up to 70ns to output data after /CAS is taken low. For a 16MHz clock signal each tick is 62.5ns. So I'd expect the gate array to be sampling the data on the next clock tick.
22. /RAMRD will only go low if the CPC is reading from built in RAM. It will not be generated by the GA if the address being read from is within a ROM which is enabled, and it will be suppressed by hardware external to the GA if an external hardware device asserts the /RAMDIS signal.
23. A read from an external RAM expansion would produce a similar trace except that the GA would generate the /RAMRD signal but it would be 'cancelled' by the expansion generating a /RAMDIS signal. This is done by a gate external to the gate array.