The great Z80 project

Written by Hans Summers
Monday, 11 January 2010 20:08 - Last Updated Monday, 11 January 2010 23:20

Introduction

The Great Z80 Computer project was originally started way back when I was 13 or so, must have been about 1984 or thereabouts. Like many of my projects, an article in an electronics magasine inspired the project. What tends to happen is that I read an article, understand something of how their circuit works, in this case how to build a Z80 computer, then my imagination gets carried away with how I might design my own. I probably bit off more than I could chew at this point... I did build some stuff and had an LED connected to the HALT light and a 8-way DIL switch on the bus. I reckoned if I gave it the HALT instruction I could get the LED to light, and indeed it did, but that's about as far as it went. Shortly after I started work on my Sidereal Clock project , which was probably a less ambitious way of learning some digital electronics (although still a hefty project!).

I hadn't even finished the sidereal clock before I started the Z80 project though. Tragically the boards are lost so I have no photographs, but I have all the circuit diagrams and will explain a bit about how it all worked, and tell the story of this long and hard struggle!

First, something about the specifications!

o Processor: Z80B (4MHz 8-bit)
o Memory: 32 KBytes RAM + 128 KBytes Display RAM
o Graphics display: 512 x 256 pixels, 256 grey scales
o Keyboard: 1980s Maplin experimenters keyboard
o Monitor: 9-inch Black & White

Later modifications were to add:

o More graphics modes, plus a text mode
o 1 MByte RAM
o 18-bit colour pallette (256K colours)

9-inch Black & White monitor by Phillips

The monitor I purchased from Build Electrical near Brighton. As you can see from the photog

{ga

The label on the top of the tube reads "Danger - Hazardous Voltage Area 12kV". This warning was probably very wise but it went for years unheeded and no case was ever built!

The great Z80 project

Written by Hans Summers
Monday, 11 January 2010 20:08 - Last Updated Monday, 11 January 2010 23:20

DANGER — HAZARDOUS VOLTAGE AREA   12kV ⚠

Circuit Diagram Explanation

The following sections describe the 10 pages of the circuit diagram. CLICK HERE to download the Z80 circuit diagram.

1. Timing generator circuit

This part of the circuit, shown on page 1, is responsible for generating appropriate horizontal and vertical synchronisation pulses for the monitor, and synchronising the rest of the display circuit so that all the right pixels come out in the right places. Standard UK television resolution is 625 lines, with a refresh rate of 50 frames per second. In reality this is slightly confusing, since each frame actually only contains 312 lines. Any one frame only draws every other line, then the next frame comes and draws the other half of the lines, which is called interlacing.

This Phillips 9-inch monitor uses a similar resolution, i.e. 312 lines which I refresh 50 times per second. This gives a line rate of 15,625 Hz, meaning each line takes 64 uS (microseconds) to draw. Of this time, a total of 16 uS is spent on the border to the left and right of the display area, and retracing to the start of the next line. Therefore the actual drawing area is 48 uS wide. I only draw 256 lines, the remaining 56 are the border at the top and bottom of the screen, and the vertical retrace. Into this 48 uS line, I squeeze 512 pixels, meaning each pixel lasts for 93.75 nS (nanoseconds). That's actually not long for the circuit to fetch the colour of the pixel from memory, output it, and get ready for the next pixel.

At the top left is the 16MHz crystal oscillator circuit, from which all timing signals for the whole computer are derived. Dividing by four gives 4MHz which is the clock signal for the Z80. Dividing some more using dual 4-bit counter chip IC2 (74LS393) gives other frequencies for use in the timing of the monitor video signal. IC4 and the gates around it derive the Horizontal synchronisation pulse, which comes once per line in the 16 uS horizontal blanking period. IC7b and IC3 count 256 lines which represent the viewable display area. When the 257'th line arrives, it resets the counters (IC3) and IC5 starts counting. IC5 and IC6 count 56 rows, which are the blank lines at the top and bottom of the picture. The gates at the bottom left generate the vertical synchonisation pulse for the monitor, somewhere in the middle of the vertical blanking period.

There's a funny story about the row counter IC3. You notice IC10a, part of a 74LS08 quad dual-input AND gate, which appears to have no useful function since both its inputs are just connected together. Well, originally I just didn't have it there at all, so the output from Q3 pin 6 of IC3 went straight to the reset pin 2. What was supposed to happen was that when the count in IC3 reached 128, it would reset itself to zero and at the same time reset the 56-row vertical-blank counter, IC5, so that it could start counting the blank rows. What actually happened is that this worked only sporadically, and as usual I was testing by watching the

signals on the monitor and scratching my head about it for ages. The picture would refuse to stabilise because the monitor had no regular vertical synchronisation pulses. What made it even worse was that when I watched the signal on the monitor, everything worked. Immediately I disconnected the monitor wire, nothing worked anymore. When I tried disconnecting the monitor from the other end of the wire, the circuit did work. So here I was with a 50 cm length of miniature screened cable, that had once been a part of a video cassette recorder (some time previously dismantled). The screen wasn't connected to ground or anything because I wasn't trying to shield anything, I just happened to be using this 50 cm length of screened cable because it was what came to hand. Now I could not work out why everything worked when the wire was there, just there, nothing connected to the other end of it, but it all stopped working when the wire was removed. Well being a practical sort of person, and not proud about such things as theoretical purity, I thought that if that was what the circuit wanted to do, that was fine by me, even if I couldn't understand why; I'd just leave the wire connected there permanently, going nowhere. I reckoned I'd curl it up small and tie it neatly to the circuit board somewhere so that everything was tidy. Only as soon as I wound it up into a small coil, it lost its healing effect and the spurious counting was back! So that didn't fix it, either. Eventually, and I have no recollection how, I worked out that what was going on was that the 74LS393 counter IC3 was resetting itself using the signal from its Q3 (pin 6) very quickly, so that Q3 only had a logic "1" on it for a few nanoseconds, and that this wasn't a stable or long enough pulse to reliably reset IC5 the 56-row vertical blank counter. The dangling wire would slow down the pulse just enough for IC5 to be able to recognise it properly, presumably by adding some capacitance and/or inductance to the Q3 output and thus blunting the pulse somewhat. So, I added the AND gate to add a 10 nS propagation delay, which turned out to be more than enough to get IC5 to reset properly and reliably as well as the 74LS393. I think this experience is an example of what is technically called a "race hazard". I learnt the hard way.Always after that, whenever I saw the chapter on race hazards in the digital electronics books I used to borrow from the library in the local town, I read them carefully and with respect.

The following diagram illustrates the timing of the horizontal and vertical sync pulses and the FRAME signal (43 lines)

A small note about the REFRESH signal at the right of the page. This is normally the same signal as the VIDEN, it is active low and signifies the REFRESH period is active. I will talk more about this later, but basically the display RAM I used is dynamic RAM. Static RAM consists of thousands of transistor flip-flops which once in their desired state, i.e. 1 or 0, stay there until they are loaded with something else. The drawback is the large number of transistors required in the memory to make all the flip-flops, which tends to mean that static RAMs are relatively small. In a dynamic RAM each bit is stored on a tiny capacitor, and you can cram a lot more of these onto a chip so dynamic RAMs have a higher capacity. At the time I made this computer, memory speed was also an issue. I couldn't get RAM chips for the 128K size I needed, at least, not without buying loads of them. But neither could I get RAM chips fast enough, remember each pixel only lasts 93.75 nS. The static RAMs I could get had access times of 350 nS or so.

So, dynamic RAMs it was. But they have a major disadvantage, which is that the capacitors are

so small, that the charge you put on them leaks away rapidly and the memory quickly forgets everything you loaded into it. In fact, this happens in only 20 mS (milliseconds). Therefore you have to "refresh" chip every 20 mS, or 50 times a second. Its not as bad as it sounds, because the memory is laid out in a square 2-dimensional matrix and you only need to refresh the rows one by one rather than every single memory cell, which is handled on chip. Since my chips were 64K x 4-bit, that meant I had to refresh 256 rows. There is no time to refresh rows while displaying the 48 uS long line, so refreshing has to take place during the 16 uS retrace period, when the display is not active. Therefore, the whole time the display is not in its viewable display area, refreshing takes place, hence the active-low REFRESH signal. As you can probably imagine, this is all tremendously complicated and fiddly to get right, which is why to the right of the circuit you can see a connector. This could be moved to the alternative position to shift the whole display 16 uS, i.e. one quarter, to the right, so that the refresh activity occured during the viewable area. Using the monitor's video signal, I could probe the circuit at various different points, and observing the patterns on the screen, I could get a fair idea what was going on with the refresh activity. It was an invaluable debugging aid I can tell you!

2. Timing logic

Moving swiftly on to the second page of circuit diagrams, we come to a whole bunch of logic circuits which look after most of control signals which are used by the rest of the circuit. This is the kind of thing many people would put inside a Programable Logic Array. There are several reasons why I didn't do that. The most important is that I had no idea what the circuit would end up like when I started. I only designed it as I went along, solving the problems as and when they came up. So I needed to be able to make changes easily. Another good reason is cost: those logic array chips were far out of my budget, particularly when you consider nearly all the chips in the Z80 project, at least the simple logic, countery type chips, were second hand from old computer boards. Plus, I didn't have any way of programming the logic arrays of course.

So what does all this logic do? Well, things are actually quite a bit more complicated than I let on when I was talking about the timing generation. What conspires to complicate matters severely is the requirement for refreshing of the dynamic RAMs. Without that requirement things would be a good deal simpler! Another quite severe problem is the speed required of the dynamic RAMs. At that time the fastest chips I could get hold of had access times of 250 nS and remember that I needed to be able to do one pixel every 93.75 nS. Each pixel has 256 gray scale levels so a whole byte gets loaded for every pixel. Clearly something had to give... Fortunately dynamic RAMs can give you a faster access time if they are used in what's called "page mode". The locations in a dynamic RAM are organised in a square 2-dimensional matrix. The row and column addresses use the same set of pins on the chip, thus halving the number if address pins required of the chip package and making the whole thing much smaller. You load a row and column one after the other by strobing low the Row Address Strobe (RAS) and Column Address Strobe (CAS) signals. In "page mode", you load one row using RAS, then keep clocking in different column addresses with CAS, to read locations in the same row. This

lends itself ideally to graphics applications because you naturally want to access memory in this sequential fashion since the video display is itself a 2-dimensional matrix of pixels.

Therefore, just before the first pixel in a row is displayed, I load the row address, then for subsequent pixels I just load the column addresses. Then I can get each pixel in a minimum of 150 nS (using the chips I had then). This still isn't good enough, its a long way from the 93.75 nS needed. So I parallel two sets of memory which I load simultaneously (2 bytes = 16 bits = 2 pixels), then I latch them and display each for half the time while the memory chips access the next location. Each memory cycle then lasts 187.5 nS. That safety margin of 37.5 nS is incredibly small, when you consider the propagation delays in TTL gates is about 10 nS and longer for counters, and that I did not use any nicely designed PCBs just plain matrix board and hand-wired connections. That is why on my Horizontal Sync timing diagram above, I have marked all sorts of timings in nano-seconds, to try to ensure that in theory all the signals would arrive at the right places at the right time.

The page-2 circuit generates these RAS and CAS signals for loading the row and column addresses into the dynamic RAM, and also the latch signals for the row and column latches on the page 3 (referred to as '373 and '374). It also generates the appropriate clock rate for the column counter (and pixel latch signal), which is actually 16 MHz divided by 3. Dividing by 2 would give 256 pulses per row, but I need 256 pulses in the displayed part of the row, which is 48 uS of the 64 uS row. In fact, this seems an appropriate time to include the diagram for page 3...

3. Memory multiplexing, row and column counters

Now, during the horizontal blanking period, refreshing occurs. At the start of the refresh period, the starting values of the column counters IC43 and IC44 are loaded with the current refresh from the refresh latch IC 45, which is a 74LS373 chip. The signal for this is "Row 193 PR" and its generated by the timing logic on page 2. The column clocks at a lower frequency during the refresh period, and only about 30 rows are refreshed in the whole 16 uS blank period. To make matters a little more confusing, remember I said that only the Rows of the dynamic RAM need to be refreshed? Well these rows are actually counted by what is in normal, display-visible times the column counter. The input to the '373 latch is just the current column count, and during the refresh the '373 just follows these inputs. Near the end of the blanking period, the refresh count stops and the '373 latches the current count ready for next time, when the column counters will again load from the '373 latch and continue the refresh. Then, just before the display of a row begins, the column counter is loaded from the octal D-type register IC 30 (74LS374). The setting in the column register is set by the processor on output port 1. Similarly the row counters IC 35 and IC 36 are loaded from the row register IC 31 at the start of each frame. The row register is on the processor's output port 2.

Think about this row and column counter presetting. It is actually a very useful feature for

scrolling the screen. Using it the processor can specify any offset coordinates for the top left pixel of the screen, with wrapping round occuring at the right and bottom of the screen. This means you can scroll the screen almost instantaneously. Compare this to the usual method which is to read every pixel from its current location and save its value to the new shifted location, which is a very time consuming task since there are so many pixels. Using this I was able to create very simple and short assembly demo programs to scroll things on the screen, which made the computer look blindingly fast compared to other small computers like ZX Spectrums and Commodore 64s. I made a short routine which made the screen black, wrote "Hans Summers" in the middle, then rotated it in a circle. The offset coordinates for the circle were stored in a lookup table, so a loop just cycled through all the coordinates. Using the > and Around the top left of the page you can see lots of 74LS157 quad 2-1 line selectors. These are required to select exactly what address actually reaches the memory. There are in fact 4 possibilities (so I could have used 74LS153s, which are dual 4-1 line selectors, but I happened to find some '157s on some old computer circuit boards so I used them). The 4 possibilities are the row count, column count, then the corresponding addresses from the CPU when it is writing to the display memory. IC12/3 select the row count into the memory, IC3/4 select the column count, and IC37/8 select the upper and lower halves of the 16-bit Z80 address but. Except for A15: the display memory only appears in the upper 32K of the Z80 address space, so only half of the 16-bit memory space can be addressed at any one time. The 16th bit A15 comes from a latch so that the CPU can select which half to address using an output port. In fact, only a quarter of the display memory is addressable at any one time, this is because remember I use a parallel set of 64 K-Byte memory to display alternate pixels so that I can fit 512 bits horizontally on a line. Selecting between the first and the alternate set of memories is also a latched bit on the output port. I originally only planned for 256 horizontal pixels, so the parallel memories are shown on another page altogether as a they were a later addition. They were also in a physically separate part of the circuit board since all the space near the original memories IC14 and IC49 was used up. I put those parallel memories IC47/8 on page 4...

4. Output ports, Z80 CPU control, refresh alarm, more memory

Page 4 has a lot of the stuff to interface the graphics display circuit to the Z80 CPU. I map 8 output ports to I/O 0-7 of the processor using the 3-8 decoder IC15. A D-type flip-flop IC16a controls CPU access to the graphics circuit. I use OUT 3 give control of the display memory to the CPU, and OUT 4 to give it back to the graphics circuit. A bit wasteful I know because I could have just used a single bit in a register on one port, but this is the way it happened. IC52b was used to clock this control so that the graphics circuit finished what it was doing before relinquishing control. I had an LED (top middle) to indicate when the CPU was communicating with the graphics board, i.e. when OUT 3 had been used. Of course, the CPU access could take only a few microseconds and certainly wouldn't usually last long enough for a flash on the LED to be seen by the human eye, so I had IC16, that old favourite a 555 timer, to generate a flash long enough to actually see. The result was every time something was written to the screen, an orange light gave a short comforting flash. IC19 is another 74LS157 and switches the memory

control signals (i.e. write, RAS, CAS etc) from the graphics circuit to the CPU control circuit so that when the CPU is supposed to be in control, it really is.

The diagram below right shows the timing of the memory control signals while the Z80 is in control. I drew this diagram at some point, from the Z80 datasheet and writing in all the propagation delays and timings in nanoseconds. I had some difficulty at that time with getting the CPU to write reliably to the display memory, so I had to have this diagram to work out which of the many signals required during a write cycle wasn't getting through at the right moment.

The dynamic RAMs must {still be) refreshed even (gather) the Z80 CPU is in control. Fortunately, the kind fo

I was a bit paranoid about all this refresh thing, so I set up an alarm circuit that would detect difficulties. You can see this on the right and side of page 4. The big NAND gate IC23 at the top right detects row 128, which resets the 555 timer IC21. As long as row 128 accesses keep coming along every so often, everything is fine. But as soon as they stop, the output of the 555 goes high and the alarm gets triggered. I used another 555, just as I did for the CPU access LED, to light a red LED for a visible amount of time when this happened. When everything was up and running it never came on, of course. But while I was sorting out the circuits to take care of refreshing, which were tricky to say the least, it was very useful to be able to see when things weren't working. Afterwards, just having that red LED there dark all the time, was comforting anyway.

5. Z80 CPU dynamic RAM timing logic

On page 5 is some stuff which overflowed from the previous 4 pages: logic to generate the dynamic RAM timing signals from the Z80 CPU when it is in control of the display memory. There is also a state register on OUT 0 which was used to set various control signals for the graphics board.

6. CPU, keyboard interface, program RAM, reset

 Here is the Z80 itself! All the signals from the Z80 were buffered because I worried that all the other circuits may drain too much current from the Z80. At the top of the page is the keyboard reading circuit. The 4 address lines A0-3 go directly to the keyboard and are decoded by a 74LS154 4-16 line decoder, which is actually in the keyboard rather than on the main computer circuit board: this just cuts down on the number of wires I have to take to the keyboard. IC70 is a 74LS244 octal 3-state buffer, enabled by IN port reads in the address range 16-31. The keyboard is connected in a matrix between the 4-16 line decoder outputs and the '244 buffer

inputs, so that effectively each input port in the range 16-31 reads the state of 8 keys of the keyboard. This makes a very simple keyboard reading circuit, and all the complicated things like debouncing were done in the software. For the Z80 program RAM I used a 62256 32K static RAM, which was mapped into the lower 32K of the Z80's 64K memory space (the display RAM was in the top 32K). At the bottom of the page is my reset circuit. On power up it generates a system reset lasting about half a second. It also controls hosting of this computer via a switch you can see at the bottom left. This switches on BUSRQ which causes the Z80 to relinquish control of all its buses. I used a Sinclair ZX Spectrum computer to write some demo routines in assembler, and then downloaded the machine code to the Z80 computer via a parallel link. An small interface board plugged into the edge connector in the back of the spectrum, and when the switch was set in the BUSRQ position, the Spectrum had control of the Z80 computer, so could download programs to its memory. It could also take control of the graphics circuit and write to the display, which was useful while testing.

7. Extra Graphics modes

Page 7 shows the additional graphics circuits I was last working on on this computer. Recall that the full graphics resolution is 512 x 256 pixels x 256 grey shades, so each pixel on the screen uses one byte of display memory. That leads to a nice display but is slow to draw and in some instances more speed would be nice. For instance, each textual character, which is plotted on an 8 x 8 pixel grid, requires 64 bytes to be written. This can really slow the processor down if its trying to draw a screen of text. My later additions (see below) were designed to address this issue, but before that I planned to modify the graphics circuit so that you could use different graphics modes. 2 pixels per byte gives 4 bits per pixel or 16 grey shades; 4 pixels per byte is 2 bits per pixel which is 4 grey shades; finally 8 pixels per byte gives a black and white display which is fine for text and much faster to draw (8 times faster).

8, 9, 10. Text board

The next 3 pages show a totally new graphics circuit which contains a character ROM. This holds the bit pattern required for each character and makes for very very fast text writing.

Homebrew Z80 computer by Marton Kun-Szabo

Don't miss this fantastic [Homebrew Z80 computer](#) built by Marton Kun-Szabo

The great Z80 project

Written by Hans Summers
Monday, 11 January 2010 20:08 - Last Updated Monday, 11 January 2010 23:20

(Hungary).

The great Z80 project

Written by Hans Summers
Monday, 11 January 2010 20:08 - Last Updated Monday, 11 January 2010 23:20

(Hungary).