DIGITAL FRAMESTORE

This project definitely rates the tag 'experimental'; and with the necessary ADC at around £100, and 48 64K DRAMs included, it isn't going to be cheap either! However, we think that our readers will be interested in the techniques involved. Design by Daniel Ogilvie.

A framestore is a device that can capture an entire image from a TV screen and freeze it electronically. The captured image can then be manipulated or combined with or compared to others.

Using a framestore, one TV camera could be used to fade between two images; one would be captured on the framestore, then the TV camera pointed at the second. A video mixer would be used to fade from one image to the other. More complex effects could be achieved by using the same image for both, with minor changes or manipulations between the two images.

Storage of the image can be synchronised with one-off events; for example, it could be synchronised with a flash gun going off; the flash is synchronised to occur during the field blanking (flyback) time and the resultant image is read into the store on the next frame scan.

A technique known as target integration can be used; here the electron beam in the camera is shut off and a feint image is built up over a period of time on the target, then the beam is turned back on again to read it into the framestore. This is similar to long exposure photography — and uses of this include astronomy.

In the design described here, the image is stored in digitial form, which makes it possible to analyse or manipulate the picture using a home computer; this will be discussed at length in a future article.

Camera tubes are available which can see into the infra red or the ultra violet, which enables us to extend our view of the world beyond conventional visible optics. For example, finger prints can be viewed under UV light, stored in the framestore, enhanced by computer and then compared with a library of finger prints for a match. Also, inks can be made to be luminous in the infrared making it possible to check for cheque or passport forgeries.

Some Television Fundamentals

Most readers will be familiar with the conventional television system used in this country which is raster scanned. The information on the brightness of the camera lense's field of view is encoded in a serial form and superimposed on synchronizing pulses which enable it to be easily recovered. Looking at your television screen, the trace starts at the top left hand corner moves across horizontally until it reaches top right where it resets back to the left, a little bit down, and scans across again.

The time taken for the horizontal line scan is 64*us* and this is known as the line period. The line scan is performed 312½ times, until the trace reaches the bottom of the screen when it returns to the middle top of the screen. The trace performs a further 312½ line scans, filling in the gaps between the first 312½ line scans, see Fig. 1.

Each set of 312½ line scans is called a *field* and requires 312½x

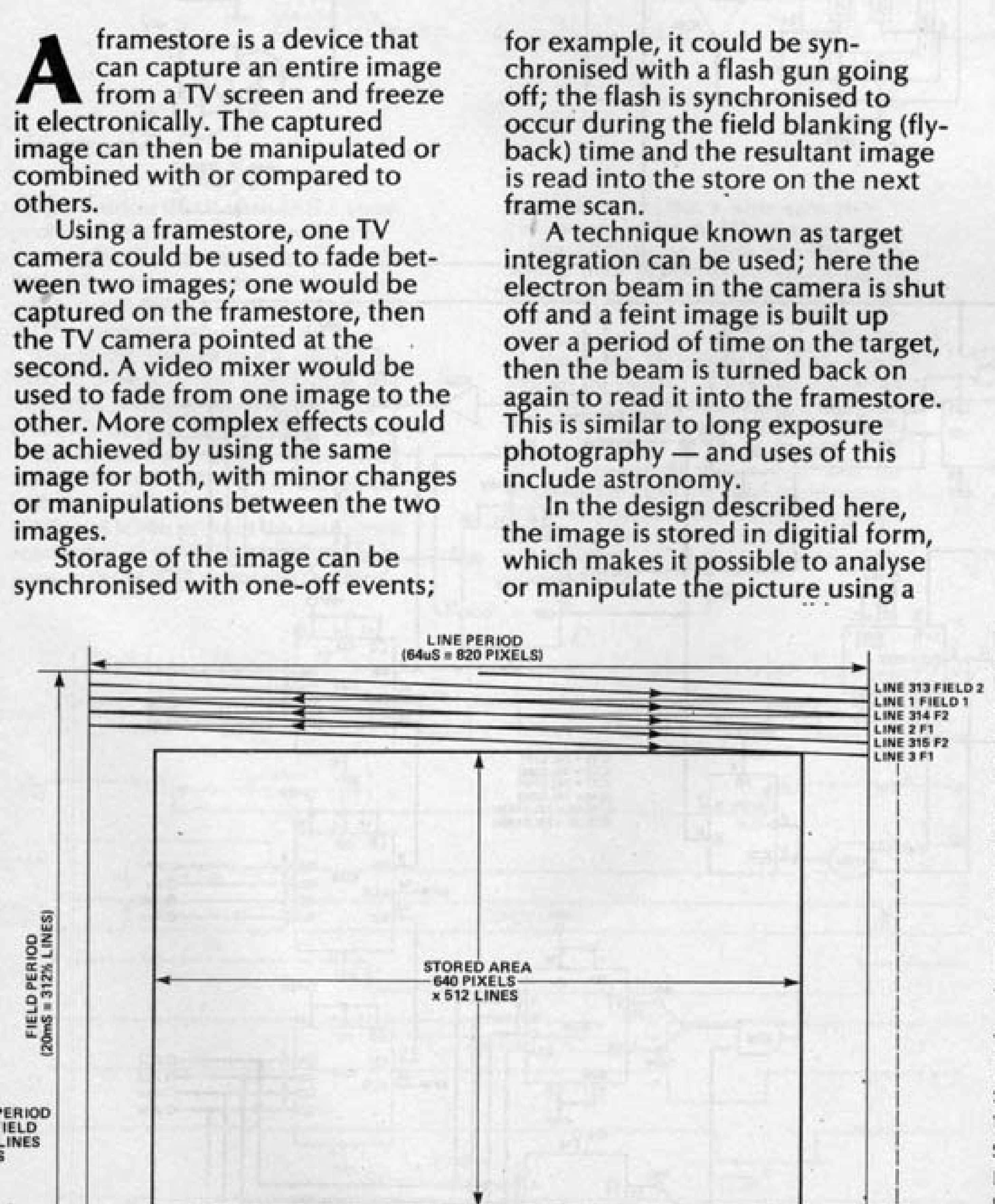
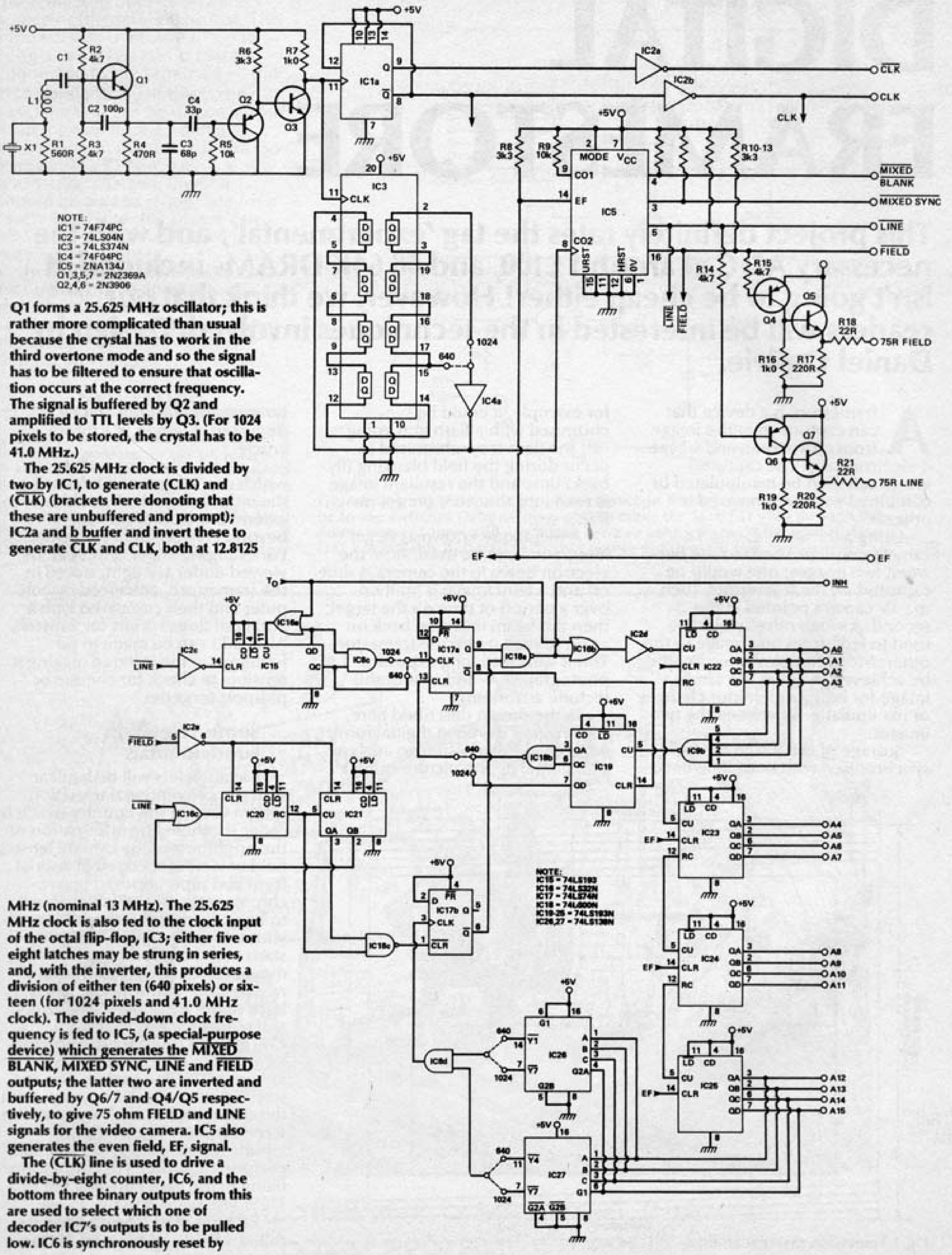


Fig.1 Television raster scanning.



Q1 forms a 25.625 MHz oscillator; this is rather more complicated than usual because the crystal has to work in the third overtone mode and so the signal has to be filtered to ensure that oscillation occurs at the correct frequency. The signal is buffered by Q2 and amplified to TTL levels by Q3. (For 1024 pixels to be stored, the crystal has to be 41.0 MHz.)

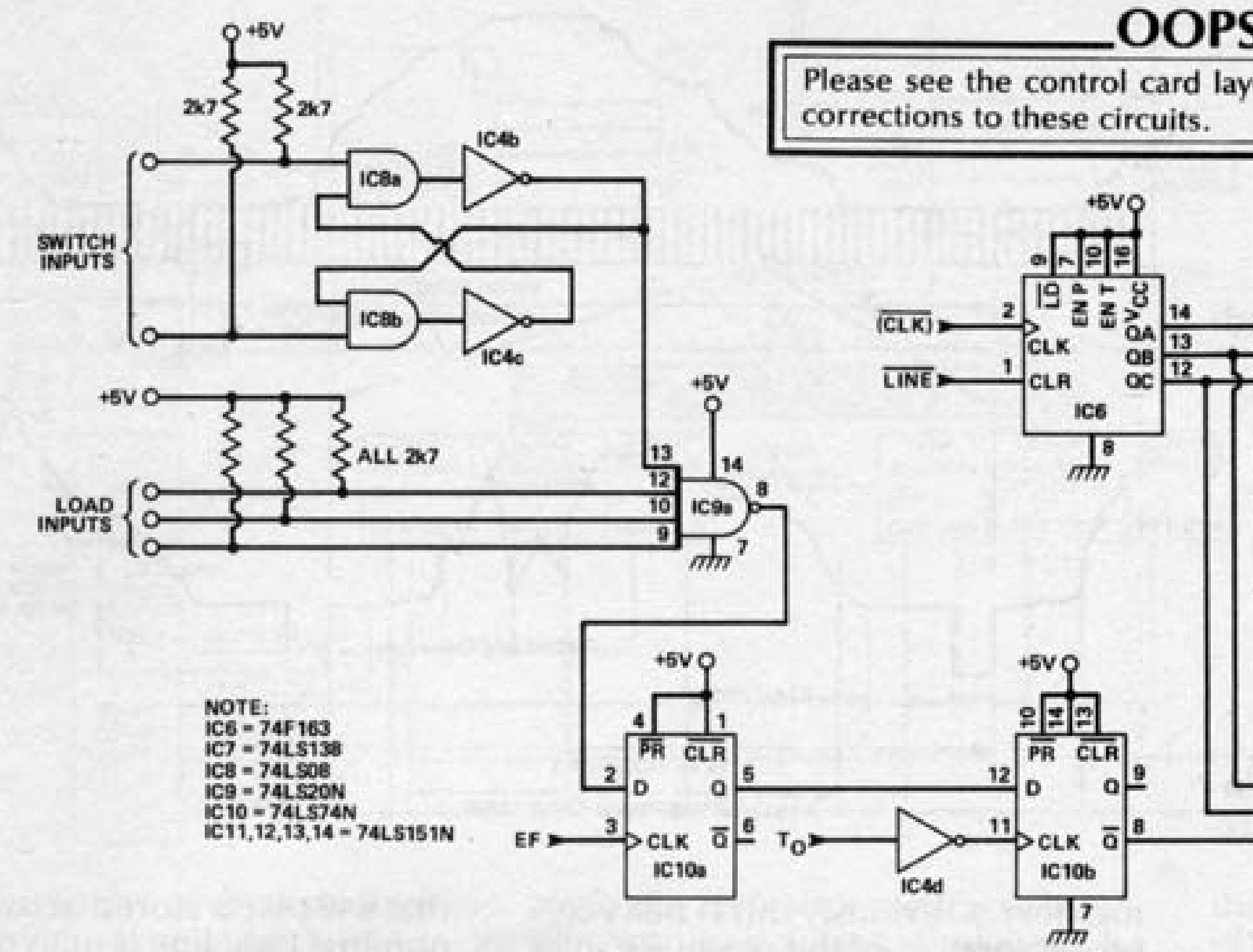
The 25.625 MHz clock is divided by two by IC1, to generate (CLK) and ($\overline{\text{CLK}}$) (brackets here donoting that these are unbuffered and prompt); IC2a and b buffer and invert these to generate $\overline{\text{CLK}}$ and CLK, both at 12.8125 MHz (nominal 13 MHz). The 25.625 MHz clock is also fed to the clock input of the octal flip-flop, IC3; either five or eight latches may be strung in series, and, with the inverter, this produces a division of either ten (640 pixels) or sixteen (for 1024 pixels and 41.0 MHz clock). The divided-down clock frequency is fed to IC5, (a special-purpose device) which generates the $\overline{\text{MIXED BLANK}}$, $\overline{\text{MIXED SYNC}}$, $\overline{\text{LINE}}$ and $\overline{\text{FIELD}}$ outputs; the latter two are inverted and buffered by Q6/7 and Q4/Q5 respectively, to give 75 ohm FIELD and LINE signals for the video camera. IC5 also generates the even field, EF, signal.

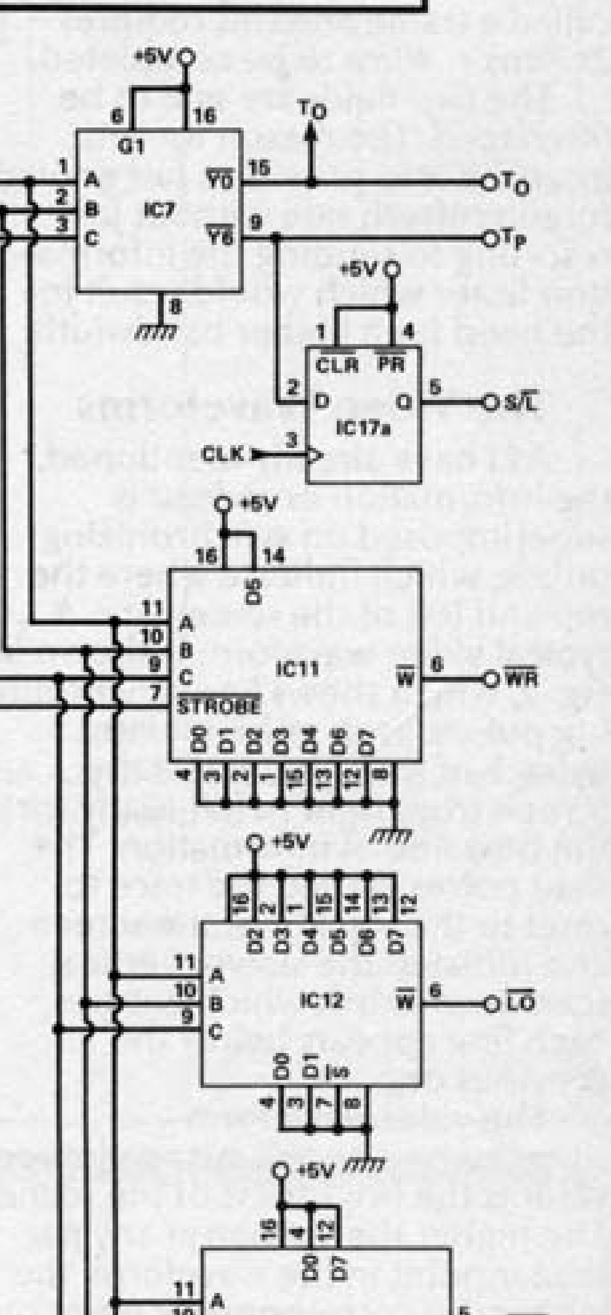
The ($\overline{\text{CLK}}$) line is used to drive a divide-by-eight counter, IC6, and the bottom three binary outputs from this are used to select which one of decoder IC7's outputs is to be pulled low. IC6 is synchronously reset by

CONTROL CARD

OOPS!

Please see the control card layout for some minor corrections to these circuits.



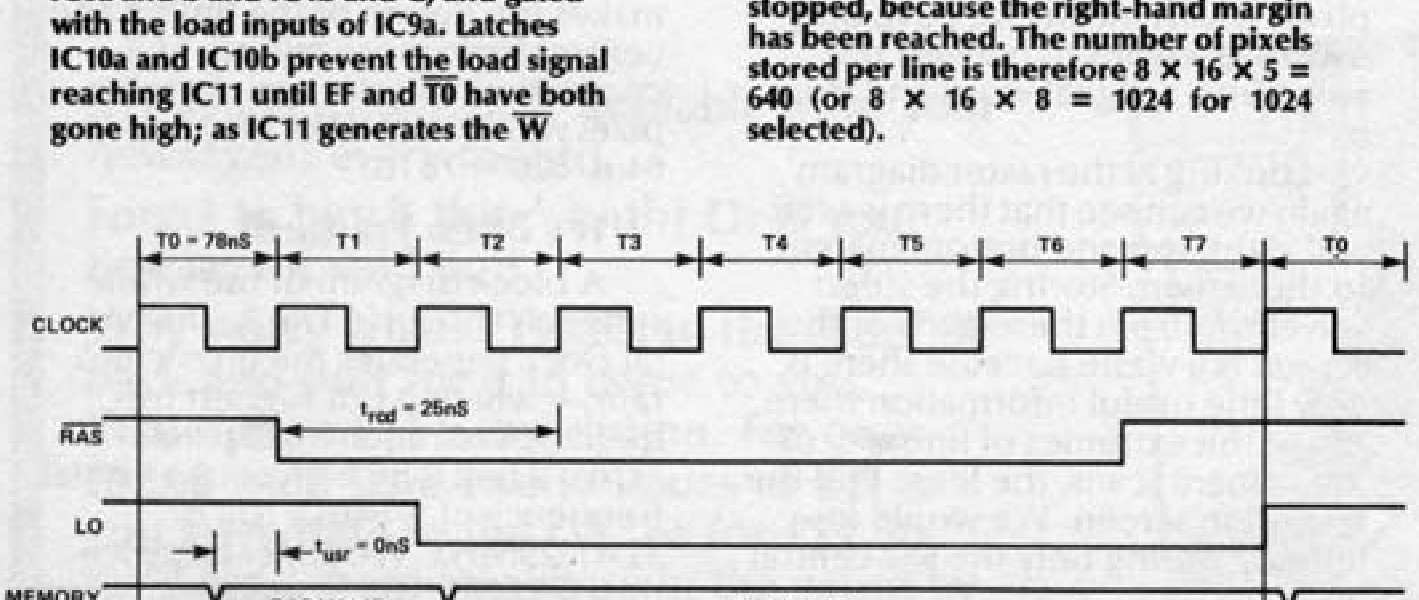


$\overline{\text{LINE}}$ to ensure that it starts in the same position every time. The $\overline{\text{Y0}}$ output is used as the timing reference, $\overline{\text{T0}}$, for the first of the eight memory access cycles while $\overline{\text{Y6}}$ provides the transfer pulse, $\overline{\text{TP}}$, for the memory buffer section.

The binary outputs of IC6 are also used to drive four data selectors, IC11-14, which generate the $\overline{\text{WR}}$, $\overline{\text{LO}}$, $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ memory control lines.

The switch inputs are de-bounced by IC8a and b and IC4b and C, and gated with the load inputs of IC9a. Latches IC10a and IC10b prevent the load signal reaching IC11 until EF and $\overline{\text{T0}}$ have both gone high; as IC11 generates the $\overline{\text{W}}$ line, this means that a write cannot commence in the middle of a frame.

The $\overline{\text{LINE}}$ pulse resets counter IC15 which then proceeds to count six $\overline{\text{T0}}$ pulses, to generate the left-hand margin (78ns × 8 × 6 = 3.74 us), which then allows the sixteen-bit counter, IC24-27, to count the $\overline{\text{T0}}$ pulses via IC1b. Every sixteen $\overline{\text{T0}}$ pulses, all the outputs from IC24 will go high; this is gated by IC9b. When this has occured five times (when 640 is selected) or eight times (when 1024 is selected) the count will be stopped, because the right-hand margin has been reached. The number of pixels stored per line is therefore 8 × 16 × 5 = 640 (or 8 × 16 × 8 = 1024 for 1024 selected).



The top and bottom margins are generated by ICS 20 and 21, which count the LINE pulses; when 32 line pulses have occurred, the flip-flop IC17b is cleared va IC18c; this takes its Q output high, allowing IC18a output to go low, so that $\overline{\text{T0}}$ pulses can pass through IC16b and be counted; so the address counters can only increment while IC17b remains cleared.Decoders IC 26 and 27 monitor the top four address lines to provide a line counter. When the counters reach 4x4906x8/640 counts = 256 lines the Y4 output of IC26 clocks IC17b via IC8d which sets this latch and halts the count until the 32 lines of the second field have elapsed when IC27 performs similarly (9x4096x8/640 counts = 512 lines). The counters are reset by the EF (even field) pulse which indicates the start of the new frame.

64us =20ms to be completed. The two fields are related in that the information in the second field augments the information in the first. The two fields together are called a frame and this requires 2x20ms = 40ms to be completed.

The two fields are said to be *interlaced.* The reason for this interlace is to provide a fast enough screen refresh rate without just resorting to sending the information faster which would result in the need for a higher bandwidth.

The Video Waveforms

As I have already mentioned, the information broadcast is superimposed on synchronizing pulses, which indicate where the top and left of the screen are. A typical video waveform is shown in Fig. 2, which shows line synchronizing pulses. Each pulse initiates a flyback of the trace across the screen from right to left ready for the next line of information. The field pulses caused the trace to reset to the top left of the screen and initiated the slower vertical scan downwards which ensures each line appears below the previous one.

The video waveform superimosed on the sync pulse represents the brightness of the scene. The higher the voltage at any particular point in the waveform, the whiter the corresponding point on the screen. The set-up voltage is defined as black and is about 45mv above the 'back porch' of the line wave-form (see Fig. 2.). The sync pulses are, therefore notionally, darker than black and this ensures that the flyback is not visible on the screen. We derive a pulse during the period of the back porch to clamp the incoming video to black to ensure we obtain a stable grey scale to our stored picture.

There are a number of other features regarding the video wafeform that will concern us, but we will deal with these as we need to.

We are concerned with the storage of one frame of this information. The method we shall use is to convert the TV screen into little packets of information and store them into a digital memory as values representing the brightness of the scene.

Memory Needs

With 625 lines to store, if we stored only one byte representing the average brightness across each line, we would require 625 bytes of memory. Obviously this is not very representative of the scene we may be looking at.

Well, let's store 625 values across the line and see what that requires, assuming that each value stored will be in the form of an eight-bit byte. We now need enough memory to store 625 lines of 625 elements (the elements are called *pixels*); this requires 625^2 bytes, ie 390,625 bytes: rather a lot!

There is an additional consideration. The line duration is 64us, and we want to break this up into 625 pixels which means we have only 64us/625=120ns to convert the video into a digital word and store it.

Looking at the raster diagram again we can see that there is a left and right, top and bottom margin to the screen. Storing the video waveform from these parts of the screen is a waste because there is very little useful information there. It is at the extremes of linearity of the camera scans, the lense and the television screen. We would lose little by storing only the 512 central lines.

Similarly the line scans contain little of use at the edges, so we need only store the central section of the lines. The screen is wider than it is deep (it has a 4:3 aspect ratio), so to maintain a similar horizontal resolution to the vertical resolution, we will need to store more than the 512, say around the 625 pixels first envisaged. Actually, the generation of the clock signals is eased considerably by prudent choice of numbers (you'll have probably already noticed the 512 lines!), and in practice it was found that 640 pixels stored across a nominal total line length of 820 pixels (much of the residue being taken up with sync and fly-back) worked out reasonably neatly. Putting these numbrs together, we arrive at a memory store requirement of 640 x 512 = 327, 680 bytes.

The memory requirement has been reduced to some extent, but we have increased the speed requirement of the A-to-D converter and of the memory. This is unfortunate because speed is costly in both these areas. It also makes the design of the timing and control logic more critical. The conversion and storage of each pixel will have to take place in 64us/820 = 78 ns.

It's Been Framed!

A block diagram of the whole system is shown in Fig. 3. The crystal clock generates the pixel clock rate — which is our highest frequency. We require 820 pixels across a line which gives us a crystal frequency of 1/64us x 820 or 12.81225MHz. We generate twice this, (25.625MHz) to ensure an even square wave master clock waveform. This is sent through the lower address counters which divide the 12.8125MHz by 820 and generate a binary address for the memory. The output from this counter will be at line frequency; the left and right margins are also generated by these counters.

The line frequency is fed to the upper address counters which divide the line frequency by 625 to provide the upper address lines to the memory, and also generate the top and bottom margins. At the

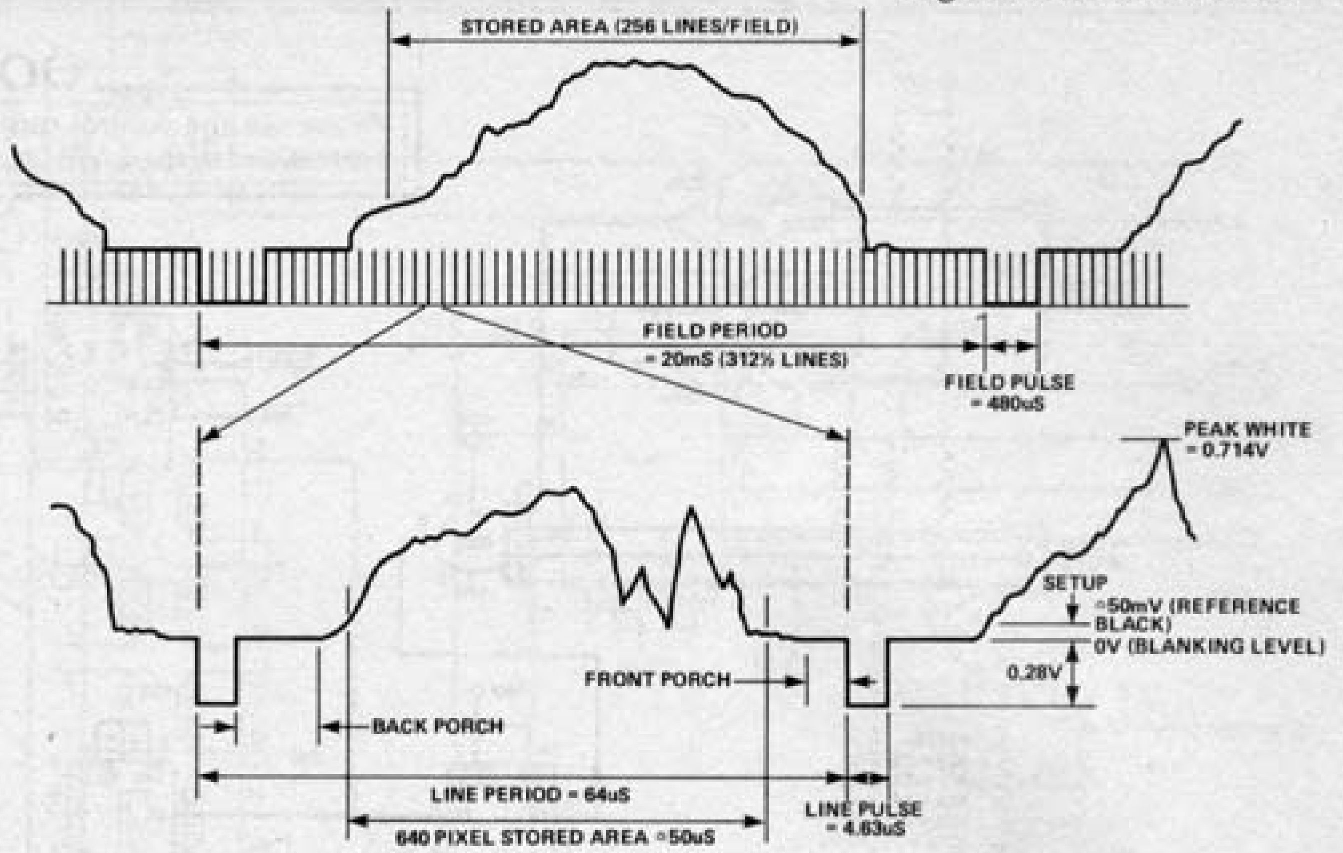
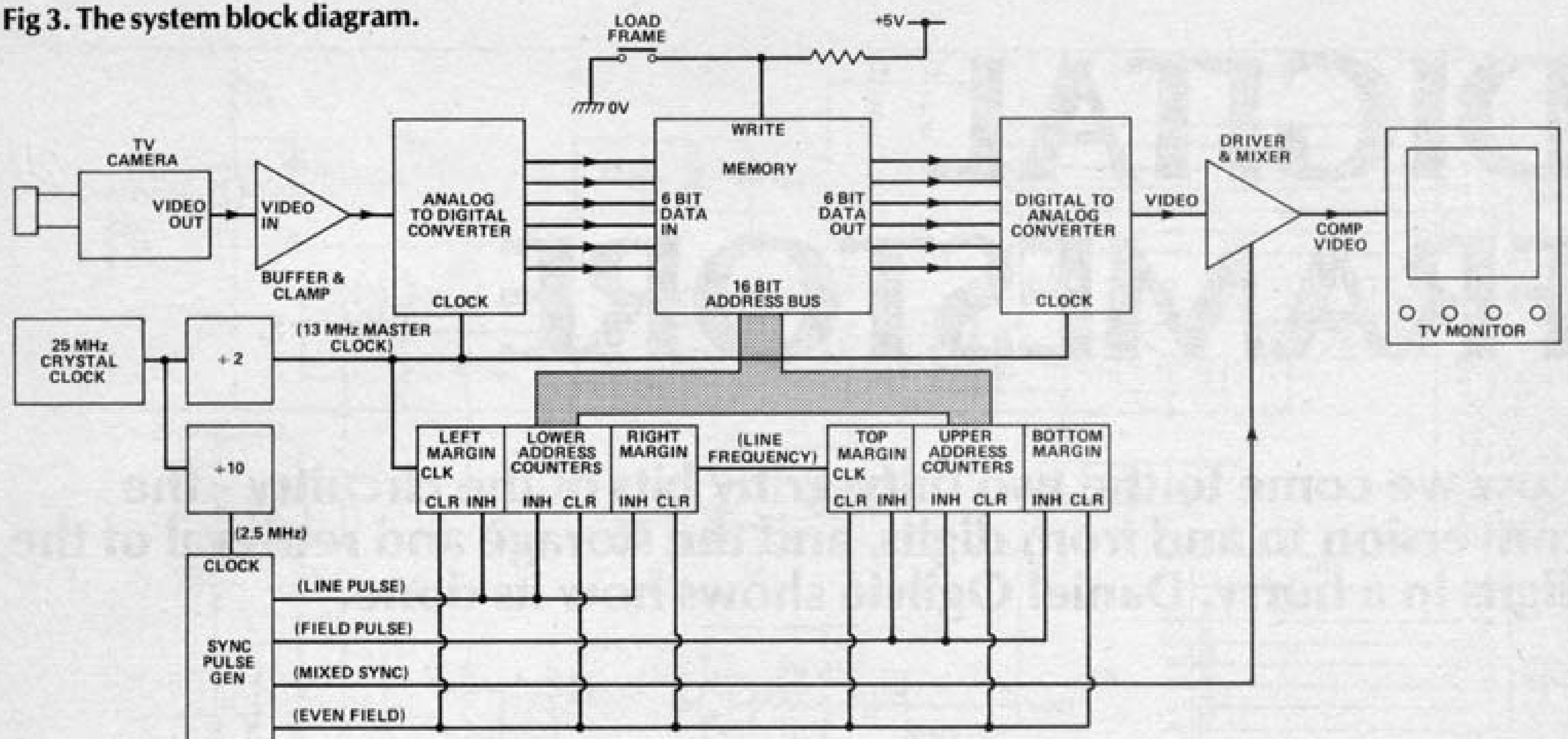


Fig 2. Video waveforms.

Fig 3. The system block diagram.



end of a frame the address counters are reset by the sync pulse generator and the process continues again.

Most of the time, we will be using the framestore to read out of memory; while this is happening, data is being sent from the memory to the DAC for conversion back to analogue and, after mixing with synchronising pulses, display on the monitor.

To store video, from, for example, a video camera, the write line on the memory is held down for the duration of one frame, while the input signal is converted by the ADC. As already mentioned, the ADC will be converting at a rate of 12.8125MHz.

To be continued ...

DIGITAL FRAMESTORE

Now we come to the two nitty-gritty bits of the circuitry - the conversion to and from digits, and the storage and retrieval of the digits in a hurry. Daniel Ogilvie shows how its done.

With a clock rate of 13MHz we have 1/13MHz = 78 ns to convert the data and store it in memory. Of the readily available ADCs, the National Semiconductor ADC0800 takes 10 *us*; obviously, we will have to look for something rather more exotic for the ADC here.

Most common ADCs work by the successive approximation technique, which requires a number of clock cycles to obtain a digital representation of the incoming analogue signal. Even at high frequencies, the number of clock cycles would take an unacceptable length of time; for instance, if we were able to clock the ADC0800 at 10MHz, it would still require 40 clock cycles to complete a conversion, which would mean that we would have to allow 4µs for each conversion — still far too slow.

Successive approximation ADCs contain just one comparator, see Fig. 4, and what they do is to try the different bits in the latch, starting with the most significant and working down to the least significant, to see which should be on in the final result.

An alternative technique is used by flash (or parallel) converters; here, there are lots of comparators all tied to the input and to different points in the resistance ladder. The encoder logic has to decide which is the highest comparator which is on. The time taken for conversion is just the propagation delay of the comparators and the encoder. However, the big disadvantage of these ADCs is the complexity, as there has to be a comparator for every possible output word. So, for an n-bit converter, there have to be 2^n comparators, and this also means that there have to be more resistors in the chain, and that the encoding logic has, necessarily, to be that much larger.

There are a number of flash converters on the market, the Ferranti ZN440 and the RCA CA3300 for example. The type we are going to use is the TRW TDC1014, which is a six-bit converter. Both eight-bit and four-bit versions are available and could be used instead; however, the four-bit version will not offer sufficient resolution for any serious application and the eight-bit version is very expensive, in effect containing four times the logic and comparators of the six-bit version. The six-bit version is a compromise.

The ADC is the single most expensive item in the framestore. It does contain a lot of very fast logic, including 64 latched comparators, for example. It is very expensive but as we have seen there is no way round this. The framestore can, of course, be built without the ADC and used only to display images loaded by computer — not exactly a framestore then though is it?

The Dynamic Ram

The DRAM is a significantly cheaper memory cell per byte than static RAM. It would be possible to design the framestore using fast (better than 70ns) access time static RAM, which would consume over 100 16K × 1, for example. These would require a second mortgage to acquire, so if only for financial reasons the DRAM would appear to be the right choice and the 64K x 1 variant is the cheapest available.

However they do require more thought and circuitry to drive them. The problem of their slow access time is discussed below. We will consider briefly here how to get data in and out of them.

You have probably noticed that the 64k × 1 DRAM comes in a 16-

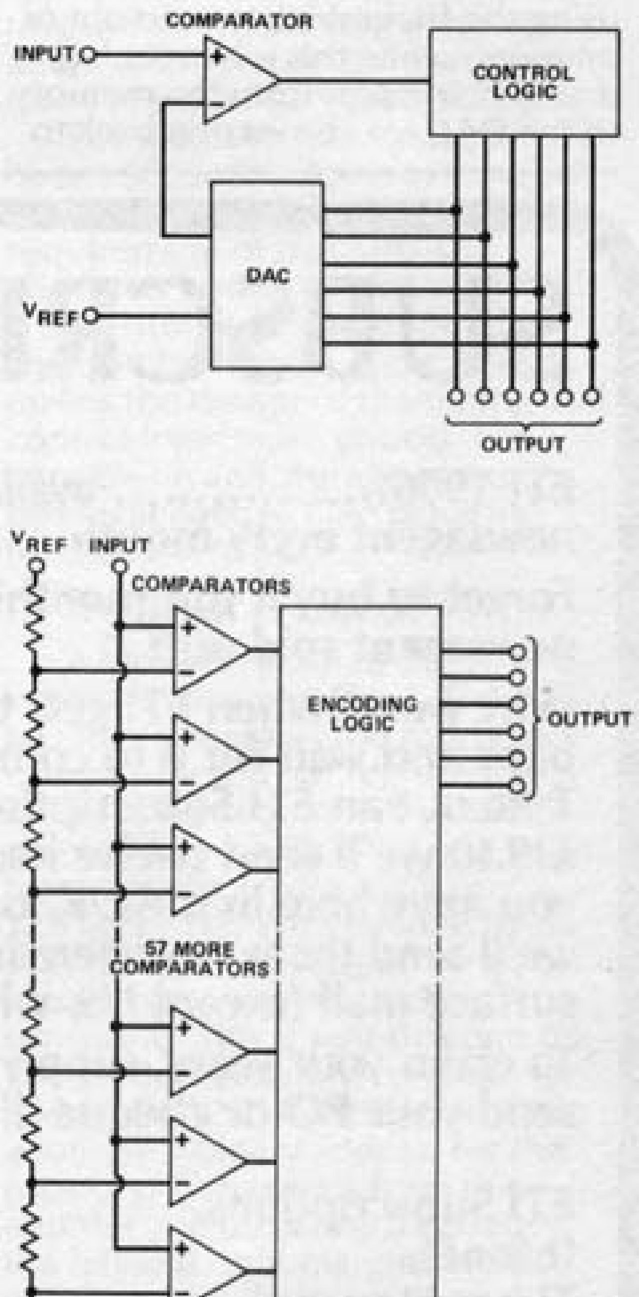


Fig. 4 Two ways of converting analogue to digital: (a) sucessive approximation ADC; (b) 'flash' converter ADC.

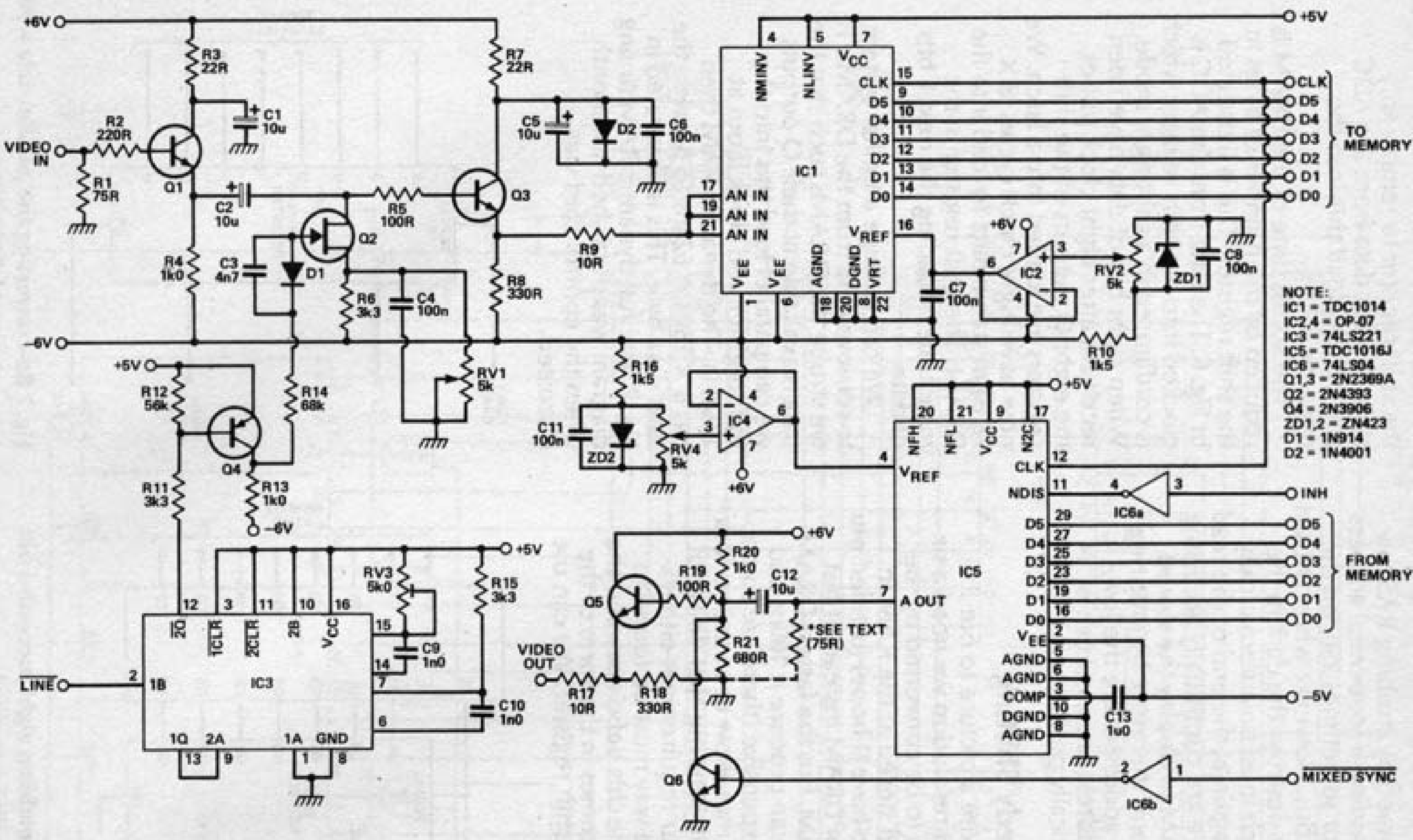


Fig. 5 Circuit diagram of the converter section.

HOW IT WORKS - ADC/DAC

R1 terminates the video signal at the appropriate 75 ohms. Q1 buffers the video, and the signal is then AC coupled to driver transistor Q3 via C2 and R5. Q3 is needed because the input to IC1, the ADC, is quite capacitive (100p).

The DC level of the signal is determined by the clamping action of Q2, as follows. IC3 is a dual monostable, and it generates a 2μs pulse 3μs after the LINE signal goes low, which is every 64μs. This pulse is fed via Q4 to the gate of Q2, which turns this transistor on, shorting the negative end of C2 to the voltage set up by R6 and RV1.

The portion of the video waveform when this is happening is known as the 'back porch', and, by definition, it is reference black. So R6 and RV1 set the reference black level for the video signal. Clamping the waveform every 64us ensures that the tone of the image remains consistent across the screen.

The ADC is IC1, a TDC1014J, which converts continuously at 13MHz. 10ns after the rising edge of the clock input, the video input is latched into its comparators and compared with the voltage fed to the reference input. This latching operation means that the input does not have to be held steady while the conversion is taking place.

On the falling edge of the clock signal, the comparator outputs are fed to a 63-to-6 decoder. The outputs from the comparators essentially form a bar-graph representing the video signal size: for example, if the video input is half the reference voltage, then half the comparators will be on and half will be off. The conversion logic takes the bar-graph output and converts it into a six-bit binary word.

The binary word is latched into the ADC output on the next rising edge of the clock and the data becomes available 30ns later. There is what is known as a one pipeline delay in the output, and the ADC takes in new data whilst converting the previous data.

The reference voltage is provided by ZD1 and this is buffered by IC2. RV2 allows the reference voltage to be changed which gives some control over the gain of the ADC; the lower the reference voltage, the lower the threshold between adjacent comparators in the ADC, so the smaller the change in the input signal that is required to change from one output binary word to the next.

IC5 is the DAC, TDC1016. It acquires data from the memory on the rising edge of the clock input provided that the data has been set up 20ns beforehand. The reference input, provided by ZD2 and buffered by IC4, is used, with the digital data, to decide the size of the output voltage. The output of the DAC can be forced to 0V (black) by the NDIS input and this is done outside the stored picture area to prevent rubbish being displayed.

The output of the DAC can drive the standard 75R line, but will provide only 500mV output swing, which will give disappointing contrast on a monitor, which would be improved by amplifying the output. On the other hand, without termination, the output voltage swing can be 1 V but fast edges of the video could cause some slight overshoot, although it is unlikely that such edges would be generated by anything other than a computer-generated image.

In the circuit shown, we leave the choice of whether or not to terminate the output from the DAC with 75R up to you; the resistor required to do this is shown dotted. The black level of the output is set by the ratio of R20 to R21 and when a MIXED SYNC pulse is present, this point is shorted to 0V by Q6 to provide a composite output.

Q5 provides a 75R drive capability for the video. No low-pass filtering has been used to reconstitute the video output, as this will almost certainly be performed by the monitor.

pin package and yet would require 16 address lines to select all of its bits (2^{16} = 64K). This is because the address is sent in two eight-bit bytes and strobed into latches by two lines (row address select,RAS, and column select,CAS). You can consider the memory as a grid with two latches selecting the vertical and horizontal rows to access the bit we require. On the falling edge of RAS the row address is latched and a little later we take CAS low which latches the other address. The CAS line also turns on the output drivers and therefore some time (the access time) after CAS the data will appear on the Q output.

If we wish to write data in we must ensure the data is valid and generate a write pulse; the data is written in on the rising edge of the pulse. It will be noticed that there is a dead time between successive accesses. This is required by the DRAM to restore the memory cell to its steady state after an access, and is referred to as the precharge time. This has to be allowed for when driving DRAMS and so the total cycle time (read or write) is the RAS access time + the precharge time.

DRAM uses a capacitor to store the data bit which loses its charge over a period of time (2ms or sometimes 4ms) and to prevent us losing data, it has to be read and written over a period of time, which is called refreshing. Refresh is performed by strobing RAS low and providing a sequential address (0 - 128 or sometimes 256) on the address bus. However when DRAM is read it destroys the data in the cell which must be automatically written again at the end of the read cycle. We are continually accessing the DRAM to display the stored picture and therefore by ensuring the RAS addresses are the low-order address lines, refresh is automatically performed.

Speedy Shifting

To store a picture to our required resolution we need a six-bit word to be converted by the ADC and stored in the RAM in 78ns. We have (I hope) justified our choice of DRAM instead of fast static RAM, but the fastest DRAM that we can procure is 100ns and comes expensive (there are some DRAMs that allow us to access four-bit data in 50ns but that does not satisfy our requirement either).

Some way must be found to overcome this deficiency and the answer comes in the form of the humble shift register. We can use the shift register to temporarily store the fast data from the ADC and parallel load this into the dynamic RAM.

The access time of the DRAM is reduced by the number of bytes in the shift register. This is illustrated in Fig. 6. The data from the ADC is clocked into the shift register which is configured in a shift right mode. When eight bits of data have been received, after eight clock pulses, the eight Q outputs of the shift registers are loaded into a latch. We now have eight clock cycles (8 × 78ns) of stable data to load into the DRAM. The shift register in the meantime is loading the next 8 bits of data.

We can arrange a similar system to retrieve data from the DRAMs. If we strobe the DRAMs such that the data is available at their Q outputs, we can parallel load this into the shift register and clock it out at 13MHz. The dynamic RAM then has 8 × 78ns = 624ns to provide the next data byte. This is illustrated in Fig. 7. The full dynamic RAM timing diagram was published last month when the control card was discussed.

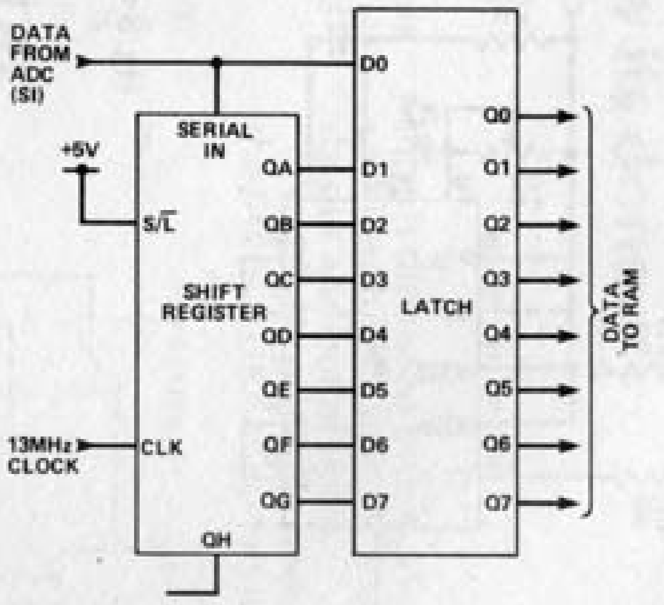


Fig. 6 Assembling eight successive bits into a parallel data word.

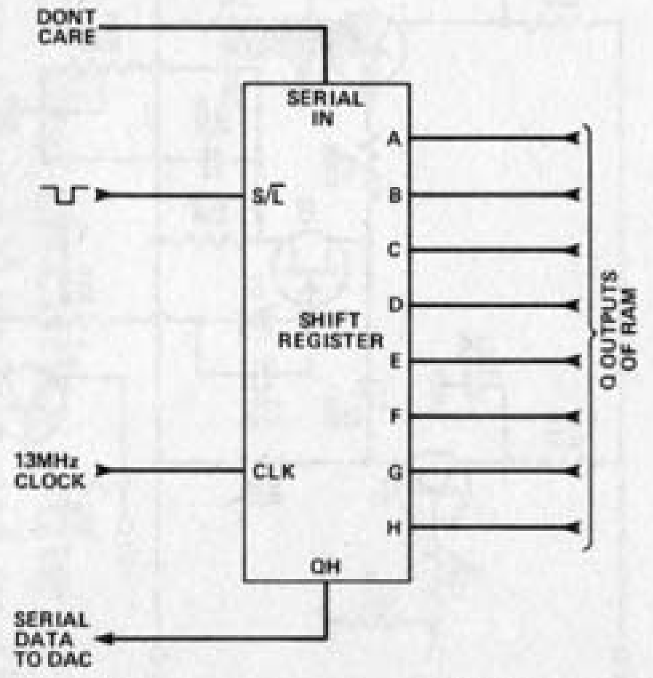


Fig. 7 Re-converting the parallel data word into a stream of eight bits.

HOW IT WORKS - MEMORY CARD

The memory chips used cannot possibly work at the speed at which the frame-store converts analogue to digital or vice versa; so the design here uses shift registers to convert eight serial bits from each of the six data bits into a parallel data word, which is then stored in parallel in memory. Thus the memory can read or write at one eighth of the speed that the ADC or DAC are working at; however, this does have the disadvantage that effectively six memory cards are required.

Fig. 8 shows the circuit for just one storage card; all the connections to this are paralleled with the other memory cards, except SI, the serial data input bit from the ADC, SO, the serial data out to the same bit on the DAC, and MPUD, which is connected to one bit of the processor data bus.

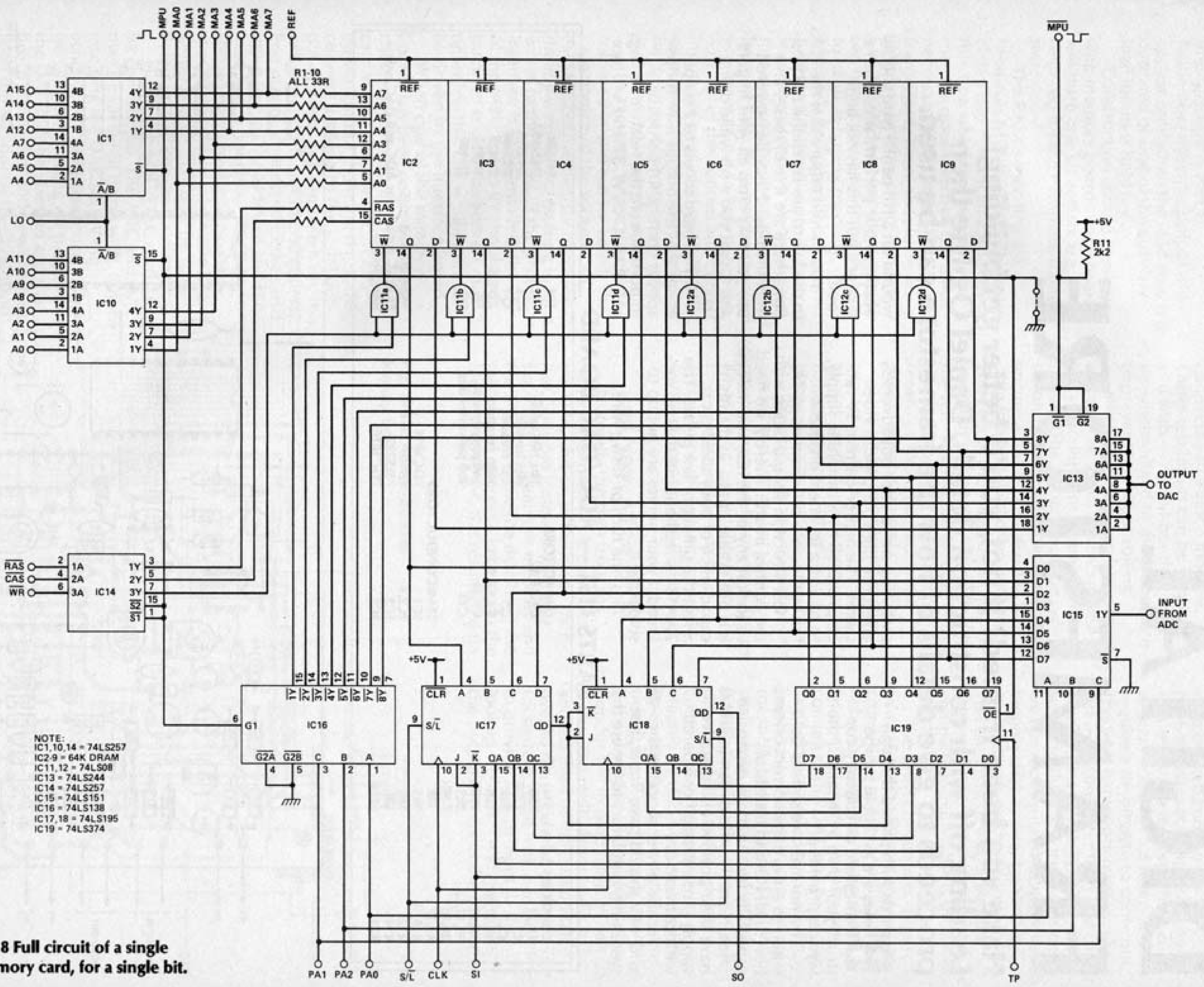
The dynamic RAM is usually being read. RAS is strobed low when the lower address bits (A0-A7) have settled. After the RAS hold time of the dynamic RAM the upper address lines are enabled and CAS is strobed low. The 16-bit address bus is multiplexed to eight bits by the two 'LS257 two-to-one multiplexers. The select line is the LO output of the control card. After the CAS access time of the RAM has elapsed the Q outputs of the DRAM become valid. These are synchronously loaded into the eight-bit data shift register formed by two 'LS195s. This eight-bit data is then clocked out of the shift register whilst the RAM is accessing the next bit.

The address inputs to the RAM have 33R resistors in series with them to damp possible destructive negative voltage excursions on the address lines due to the high capacitance of the DRAM inputs and the PCB tracks.

Should we wish to load data into the RAM we need to pull down the write line to the RAM at the appropriate time; this is handled by the control card. Data from the ADC is clocked into the shift registers. When eight bits have been loaded, the Q outputs of the shift register are loaded into an eight-bit latch on the rising edge of the transfer pulse. The data is then valid and held for the next write cycle of the DRAM.

Any additional I/O on the board is for MPU access and will be discussed in a future article.

The type of RAM shown is Motorola MCM6664. This has an on-chip refresh counter which can make interfacing to an MPU a little easier. However any 64K 200ns DRAM should work quite satisfactorily as there are no critical timings.



NOTE:
IC1,10,14 = 74LS257
IC2-9 = 64K DRAM
IC11,12 = 74LS08
IC13 = 74LS244
IC14 = 74LS257
IC15 = 74LS151
IC16 = 74LS138
IC17,18 = 74LS195
IC19 = 74LS374

Fig. 8 Full circuit of a single memory card, for a single bit.

To be continued (still) ...

DIGITAL FRAMESTORE

Have you been framed? If not, you'd better get building! Leading off with construction details, Daniel Ogilvie then proceeds to give details of how the framestore can be used.

Boards with plated through holes are available for the framestore or you may construct your own by means of the layouts provided.

The normal care should be taken to ensure the correct orientation of all ICs and transistors — especially for the more expensive ones. Note that the dynamic RAMs have their positive and negative supply pins swapped compared with convention.

For the power supplies, the −5V and +6V should have linear, not switching, regulators to ensure that switching noise does not adversely affect the high bandwidth video stages. The higher-current 5V can be provided by a switching supply, to ensure a low heat dissipation: the framestore can draw up to 4A.

Construction should begin with the control board. There is no need to socket any of the ICs, and indeed it is preferable not to, as the high capacitance of poor sockets can affect the timing of fast pulses. The ZNA134 is resilient but may be socketed if your nerves are not up to soldering it.

LI is one turn of 18swg wire wound on a HB pencil (you should obtain similar performance from a B pencil but this has not been tried).

You should ensure that the links in the board are set to the 640 position (there are five of these) and then the board may be powered up; you may cross all your fingers and toes if you wish while doing this.

If you have access to a scope check that the output waveforms appear correct. Of particular importance are the relative timings of 0, RAS, CAS, W, TP and S/L (see

PARTS LIST — ADC/DAC BOARD

RESISTORS (all ¼W 5%)	
R1,22	75R
R2	220R
R3,7	22R
R4,13,20	1k0
R5,19	100R
R6,11,15	3k3
R8,18	330R
R9,17	10R
R10,16	1k5
R12	56k
R14	68k
R21	680R

CAPACITORS	
C1,2,5,12	10u 16V axial electrolytic
C3	4n7 ceramic
C4,6,7,8,11	100n ceramic
C9,10	1n0 polystyrene
C13	1u0 polyester film

SEMICONDUCTORS	
IC1	TDC1014
IC2,4	OP-07
IC3	74LS221
IC5	TDC1016J
IC6	74LS04
Q1,3	2N2369A
Q2	2N4393
Q4	2N3906
ZD1,2	ZN423
D1	1N914
D2	1N4001

MISCELLANEOUS
PCB; IC sockets for ADC and DAC only; wire, solder, etc.

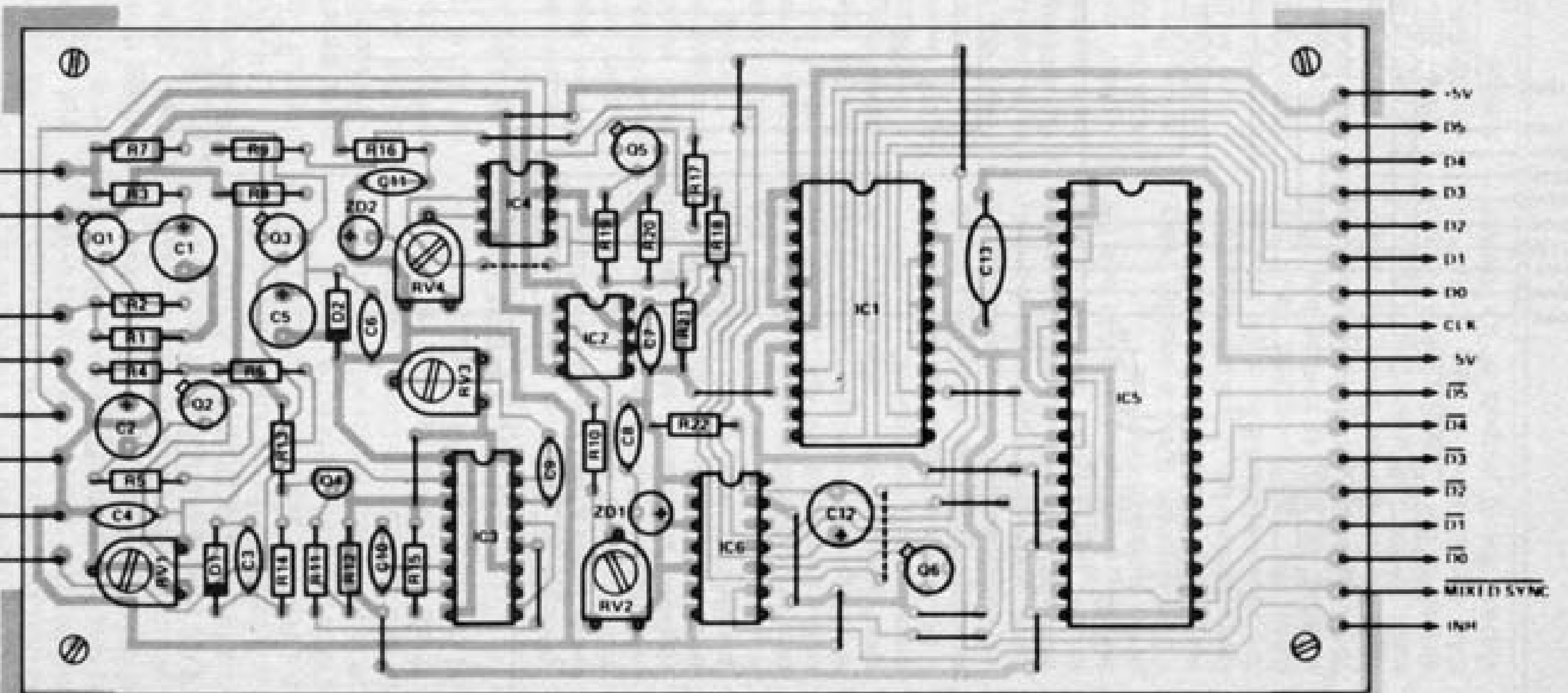


Fig. 9 The component overlay of the ADC/DAC board.

timing diagram, Part 1)
Remember the write output will appear only when the load switch is operated or one of the auxiliary load inputs is held low.

Construction should continue with memory boards. Each board will store one bit and six are required for the complete framestore.

The RAM's specified are Motorola MCM6664L20. These are cheap and contain an auto-refresh function which can be useful for the processor interface but is not necessary under normal use as a framestore. There should be no problem in using any other 200ns 64K × 1 DRAMs, but this is suggested only if you have some DRAMs to hand. If buying specially, the Motorola parts should be ordered unless a significant price difference is apparent, and we would suggest that eight of the alternatives should be tried first. They may be installed one bit at a time to help space the cost. All of the address and control inputs are paralleled up via the IDC connector. ICs 13 and 15 need not be inserted if the processor interface is not to be constructed.

The dynamic RAMs should be soldered in and not socketed: the capacitance of the sockets can cause excessive overshoot on the driving waveforms resulting in false writes to the RAMs. If the processor interface is to be installed, the link of the MPU line to 0V should be put in to enable the address multiplexers. This line would normally be driven by the interface board.

When one of the memory boards has been constructed and checked, connect it to the control card via the IDC connector and power both of them up. If possible, connect a 'scope to the serial output of the board. On switch on, this will be random, but the A0 address line may be connected to the serial input to the DRAM board and the load switch operated. The DRAM should now provide a regularly spaced waveform across one line. Other boards may now be paralleled up and checked similarly.

Finally the ADC/DAC card can be constructed. The ADC and DAC may be socketed if you wish. Before inserting the DAC set RV4 to achieve −1V at its reference input (pin 4). After turning off the power, the DAC may be inserted and the board connected and powered up. Load the A0 address line again and connect the serial output of the memory board to the MSB of the DAC.

The video output can now be connected to a video monitor. Other test inputs can be applied if you wish — remember they must be synchronized to the framestore to prevent tearing or rolling.

Finally the ADC can be inserted and the test input can be applied to the video input or a camera connected. The camera must be synchronized to the framestore to ensure a stable picture using the 75 ohm line and field drive outputs. The gain of the video input may be varied by means of RV1 which alters the reference to the ADC. An offset may be applied to the input by means of RV2. The framestore should now be up and running.

Due to the complexity of the project it is difficult to give any guidelines on fault finding should the worst occur. However to encourage contruction the author is willing to offer a trouble-shooting service and will undertake the repair of any one board for £20 plus any parts necessary (see Editorial Note at the end for details).
This will not apply to any construction that does not use PTH boards.

And Now, A Few Tricks

Normally the data from the framestores memory is passed to the DAC and on to the video monitor. Suppose instead we insert a fast RAM between the memory and the DAC, and use the data from the memory to address a location in the high speed RAM, the resultant data read being passed to the DAC. We have six bytes coming from the framestore's memory which can address one of 64 locations and we require a six-bit byte to come out. This 64 × 6 memory must also be fast — ideally faster than one of our clock cycles (78ns). We also need access to the RAM from outside so we can modify its contents; this can be provided by two, two-input multiplexors on the address inputs. This circuit is, in fact, a humble look-up table, and the circuit of the complete lookup table is shown in Fig. 10.

The RAM chosen is configured in a 256 × 9 bit format and has an access time of 45ns. Adding the delay through the multiplexors (about 5ns) means that the data coming to the DAC will be latched in one clock cycle later. This is not important, although we must delay the inhibit signal to the DAC (NDIS) similarly or we will lose the right hand column of the screen and will display rubbish in the left hand column. The required delay is created by IC2. Access to the RAM by our processor is quite conventional. The CS input from the MPU is the decoded 64-bit address location for the lookup table. When CS is low, the multiplexors switch the address lines to the lookup table over to the MPU address lines. The CS signal is rated with our R/W line to provide a write input to the lookup table, and it also disables the output drivers of the lookup

Fig. 10 The not-so-humble lookup table.

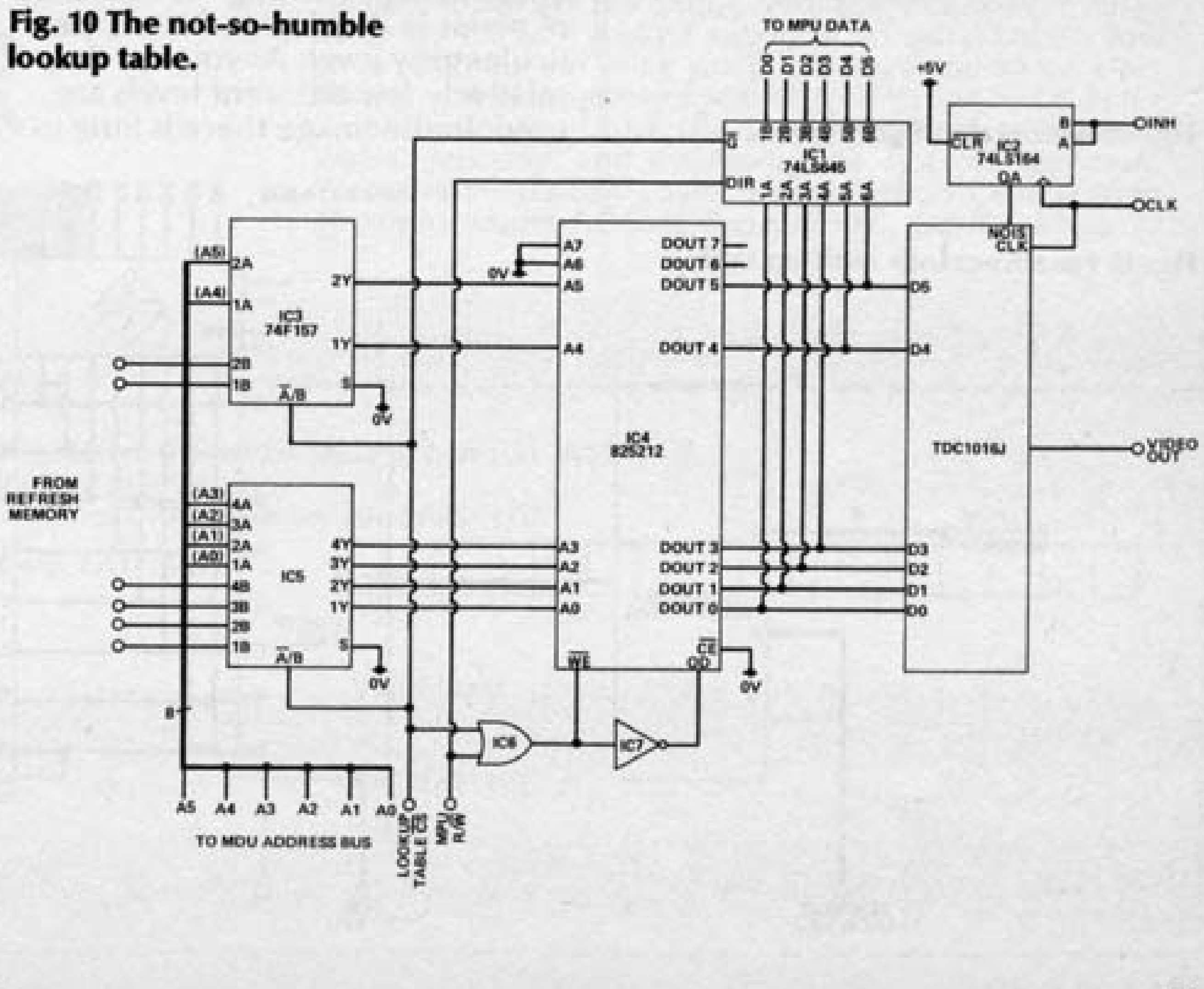


table which enables us to load data in via the tri-state buffer, IC1.

The writing to the lookup table should usually be performed during the blanking (inhibited) period of the display to prevent interference on it.

Normally the lookup table is loaded up such that a normal picture is obtained. All address locations are loaded up with a byte corresponding to the address bus value, i.e. address 0 is loaded with 0, address 1 is loaded with 1, etc, and address 63 is loaded with 63. The incoming data from the framestore memory just addresses a similar valued byte which is converted by the DAC. This gives us complete control of the grey-level structure of the image. If we want to highlight a pixel value or a range of pixels we can load the corresponding address value with peak white; for example to highlight grey level 24 we load up the lookup table address 24 with the byte 63 to turn it white.

Have a look at Fig. 11. Fig. 12, shown underneath, is a histogram of that picture which has been obtained by counting the numbers of pixels in that image for any particular grey level. As you can see, relatively few different levels are used. In the image there is little in the dark grey or black tones and little approaching peak white. Because the picture contains few grey levels, it appears flat and little detailed information can be obtained from it. We could improve this by stretching the histogram over our full 63 grey level range. The increments between the pixels are greater, of course, but more detail is apparent. The resultant histogram is shown in Fig. 13 and by loading the lookup table with the new values we obtain the

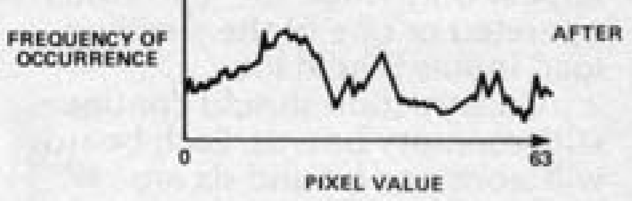


Fig. 13 Enhanced histogram.

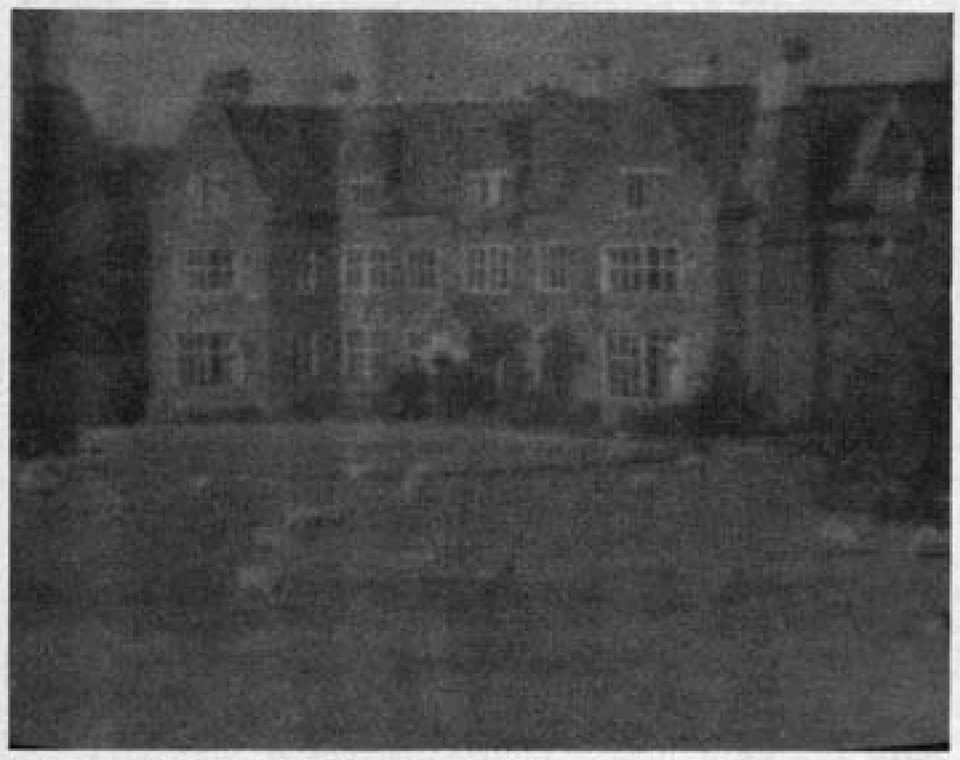


Fig. 11 Picture before enhancement.

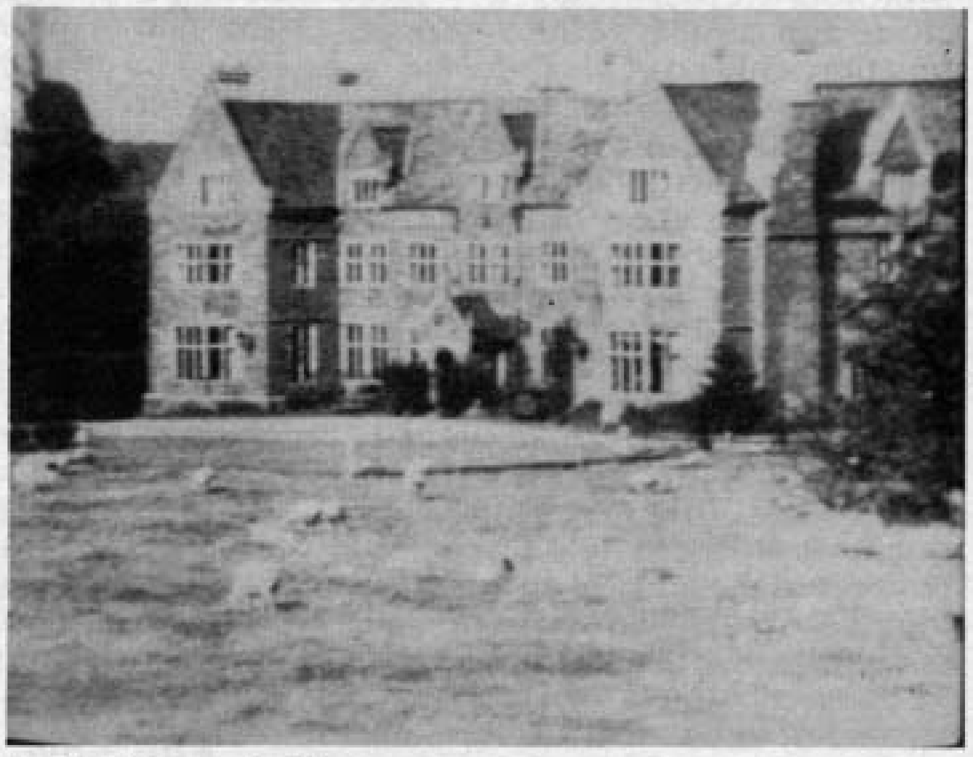


Fig. 14 Enhanced picture.

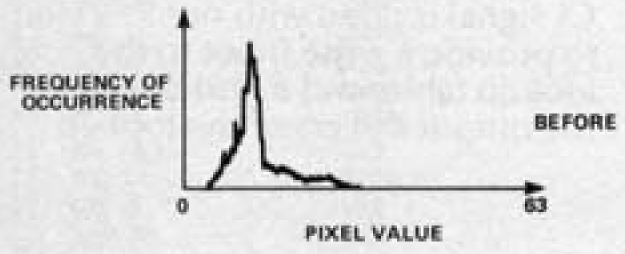


Fig. 12 Histogram of grey levels.

Fig. 15 Psuedo-colour look-up table.

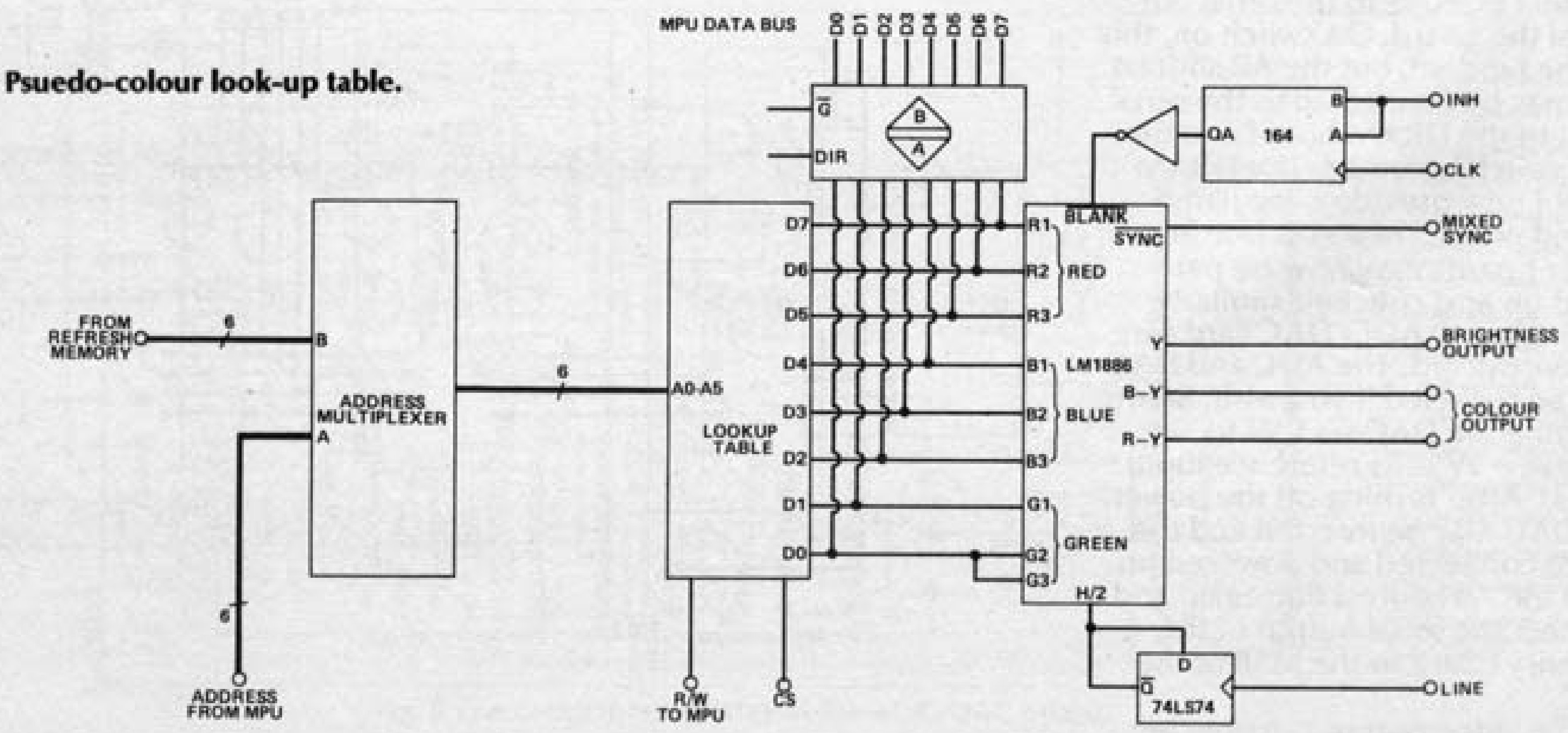


photo shown in Fig. 14. The lookup-table has only manipulated the output of the framestore and is therefore non destructive — we can restore the lookup table to its original values at any time to obtain our original image. On top of that we have only to write to 64 locations which can be done during the field blanking period and allows us to perform virtually real time image processing. One further use of the lookup table is to provide a pseudo-colour output. For example our highlighted pixel (value 24) could have been turned a different colour which might have highlighted features or differences more efficiently. The six bit output of the framestore can be used to address a video DAC with colour capability. An example of this is shown in Fig. 15, using the National Semiconductor LMI889. This IC will accept a nine-byte binary input and provide a r-y and b-y output to drive a modulator or monitor directly. Readers are referred to the National Semiconductor linear book for detailed information on the device.

A more effective (and expensive) colour display can be obtained by the use of three look-up tables and three DACs, and this is illustrated in Fig. 16. Each look-up table is dedicated to driving either the red, green or blue output. If we load each look-up table with our initial values (1 to 1, 2 to 2 etc) we will obtain our grey scale output. However we can turn the image into a green, red or blue display by writing zeros into the corresponding lookup tables (e.g. to obtain a blue image we should write zeros to the red and green lookup tables and leave the blue as it is).

In fact it can be shown that we can turn any grey level into any one of $(2^2)^3 = 262,000$ colours. We can only display any 64 of them at one time, of course.

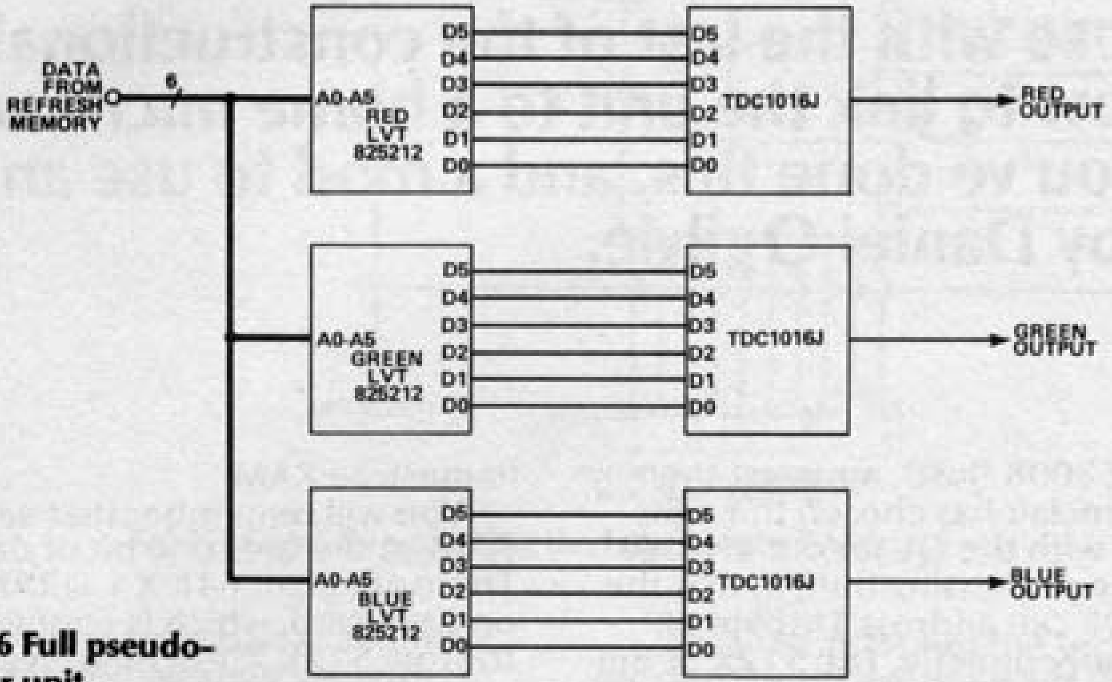


Fig. 16 Full pseudo-colour unit.

The PCB overlay for a Memory Card is published in the following part, while the Control Card overlay is given in the fifth and final part of this article. ■

DIGITAL FRAMESTORE

The project draws to a close with the last of the constructional details, suggestions on how to link the unit to a home micro and what to do with it when you've done this, and a mod to use an external sync source. All by Daniel Ogilvie.

Essentially the framestore is a large piece of memory. Various bits and bobs have been added to it to format the memory so that it can be easily written to and read from in a format compatible with a raster-scanned TV system. However at the heart of it all is a large piece (512K) of dynamic RAM just waiting to be got at by your home computer. The home computer itself can perform some quite powerful image processing routines.

We have seen that grey level manipulations can be performed by the lookup table and for the type of thing discussed that offers us a faster non-destructive method. It could equally have been performed by the home micro. Indeed, a micro with access to the framestore memory was essential to construct the grey level histogram. Image storage is another area where the home computer can be of some use, although your average floppy disk will throw a fit at having to cram on the 393K bytes necessary to store just one image. However parts of images may be stored and the more adventurous may choose to write some image compression routines, a 10:1 reduction being possible on simple images.

Getting At The RAM

Most home computers are based on either the 6502 or Z80 MPU's, with a smattering of 6809 and 68008 (just), amongst them. Mr Sinclair has chosen the right road with the QL for our average image processing buff, in that the 68008 can address 1Mbyte of memory directly. The 512K of the framestore can slot in nicely. Most micros are restricted to 64K and by the time we have added an operating system or two and some of its own RAM there may be little left to access the framestore. There is a lot to be said, therefore, for a dedicated micro providing a serial or, preferably, a faster parallel interface to the home computer, or providing a DMA interface to shift chunks of the framestore memory's data to and from the micro's own RAM.

We will not take this approach, however, but will make the assumption that at least a 16K block can be freed through which we can access the framestore RAM by bank selection. The popularity of the home micros has been reflected in two designs recently in ETI for dynamic RAM controllers for the 6502 and Z80. Also recommended is the excellent Texas TMS4500A DRAM controller user manual, which provides circuits to interface some other microprocessors to DRAM, including the 68000, 8085 and TMS9995.

Other DRAM controllers are available, from AMD and Intel amongst others, with exhaustive application notes. We will concentrate therefore on the bank select logic and particular points regarding the interface to the framestore RAM.

You will remember that each RAM card stores one bit of data. There are eight 64K X 1 DRAMs on each card, which is configured to store 512K X 1 of data. We provide an eight bit shift register on the card which temporarily stores the incoming data before we parallel load this 8-bit byte into the RAM. This overcomes the relatively slow access time of the DRAM. Each DRAM therefore stores every eighth bit of the same DRAM address.

The facility has been provided on the card to turn off the drivers to the RAM address and control inputs. This allows access to an external DRAM controller. When the $\overline{\text{MPU}}$ line is pulled low (and MPU is high) the DRAM address multiplexors and $\overline{\text{WR}}$, $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ drivers are turned off, (high impedance) as is IC17, the latch that drives the data lines to the RAM.

We now have complete access to the RAM on the card and are free to access any of the 64K bits of RAM. We do, however, have to perform some muliplexing of the data and control lines to enable us to sequentially access pixels from the DRAM and not have to worry about the complications caused by the shift register. Were we not to do this, and, for example, tied each data line of the DRAMs to a separate MPU line, sequential pixels would appear on each line of the MPU data bus.

To access any of the remaining five data bits we would need to

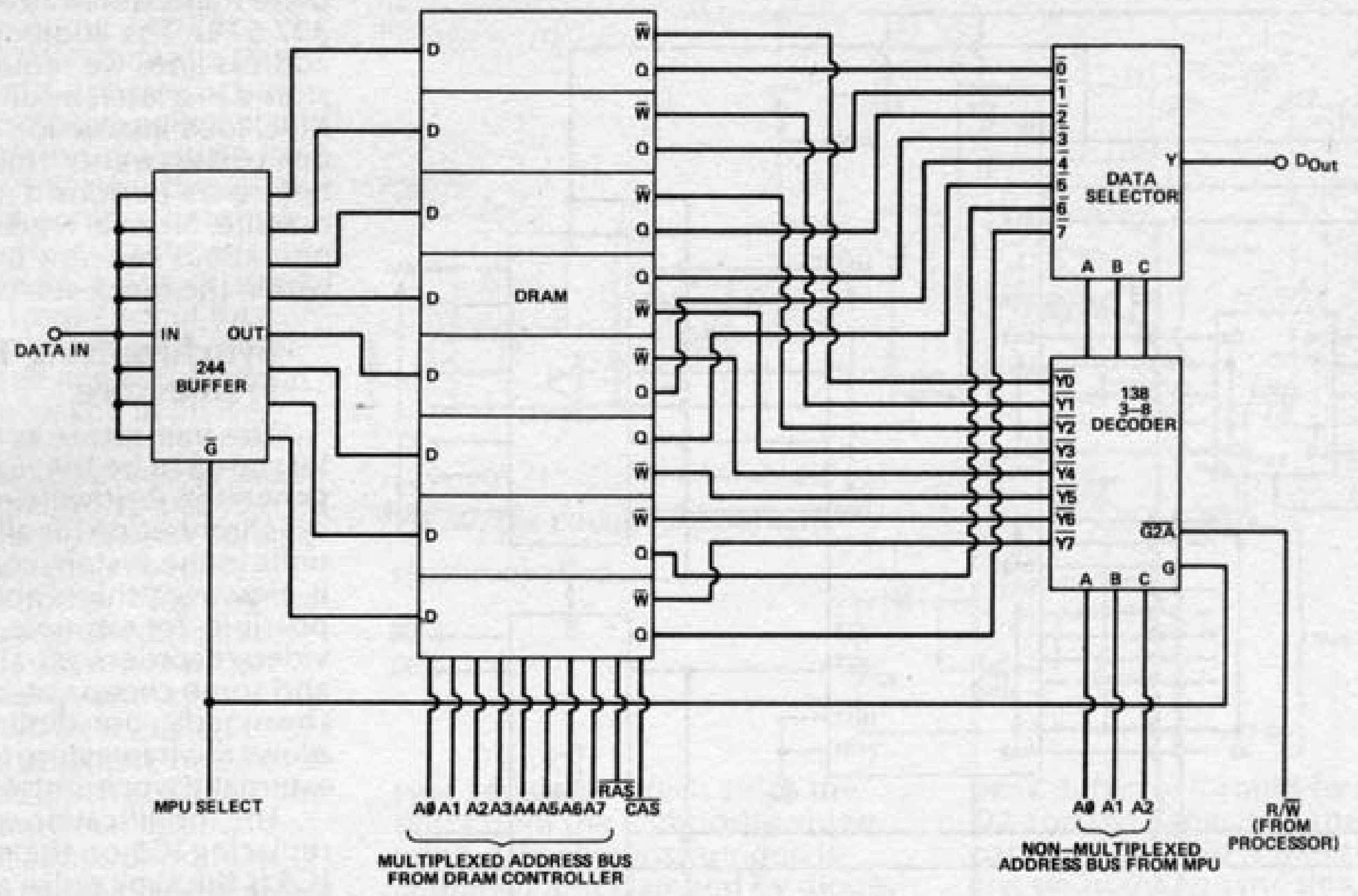


Fig 17. Directly accessing the Framestore RAM.

select a separate part of memory. For example, assume we wish to read from the framestore memory. First set up the most significant address lines of the microprocessor bus (latching them into a port), then perform a memory read operation at the address we want to access. The DRAM controller performs the muliplexing of all but the lower three address lines and then strobes $\overline{RAS}$ and $\overline{CAS}$ low in turn. When $\overline{CAS}$ is strobed low, all eight dynamic RAMs turn on their output drivers and, after the CAS access time, the data at the address we have selected becomes valid.

In fact we access eight sequential pixels worth of data at the same time. The data outputs from the DRAM are taken to the eight to one muliplexor IC19. We select one of the eight DRAM outputs by means of the three lower address lines: the data bit appearing on the MPU data bus is thus just one of the selected pixels. If we wish to access the next sequential pixel we increment the address line by one. The address loaded in to the RAM is the same but the lower three address lines select the next bit from the next DRAM. This is performed on all six boards simultaneously — each board drives a separate MPU data line — only D0-D5 of the MPU data bus are used. This process is illustrated diagramatically in Fig. 17.

This method is not the most efficient to access the RAM, but it is simple. By strobing all of the $\overline{CAS}$ lines simultaneously (and thus turning on all of the RAM drivers simultaneously) maximum current is taken. We are turning on eight RAMs to access one per board. Ideally we should multiplex the $\overline{CAS}$ lines to the RAM's using the same method we use for writing.

Writing to the RAM is performed much the same as reading. The DRAM controller responds to a write access request by strobing $\overline{RAS}$ and $\overline{CAS}$ low to latch the two eight bit address inputs. Because the R/$\overline{W}$ arrives before CAS (read/write is set up with the address lines by the MPU) the DRAMs perform an early write and the Q outputs will remain in a high impedance state. When R/$\overline{W}$ is low and a valid $\overline{CS}$ has been received, the 74LS138(IC14) decodes the lower three adddress lines and the appropriate Y output is strobed low driving the DRAM write line low, and latching in the data that has been set up on the D inputs (and buffered by IC18).

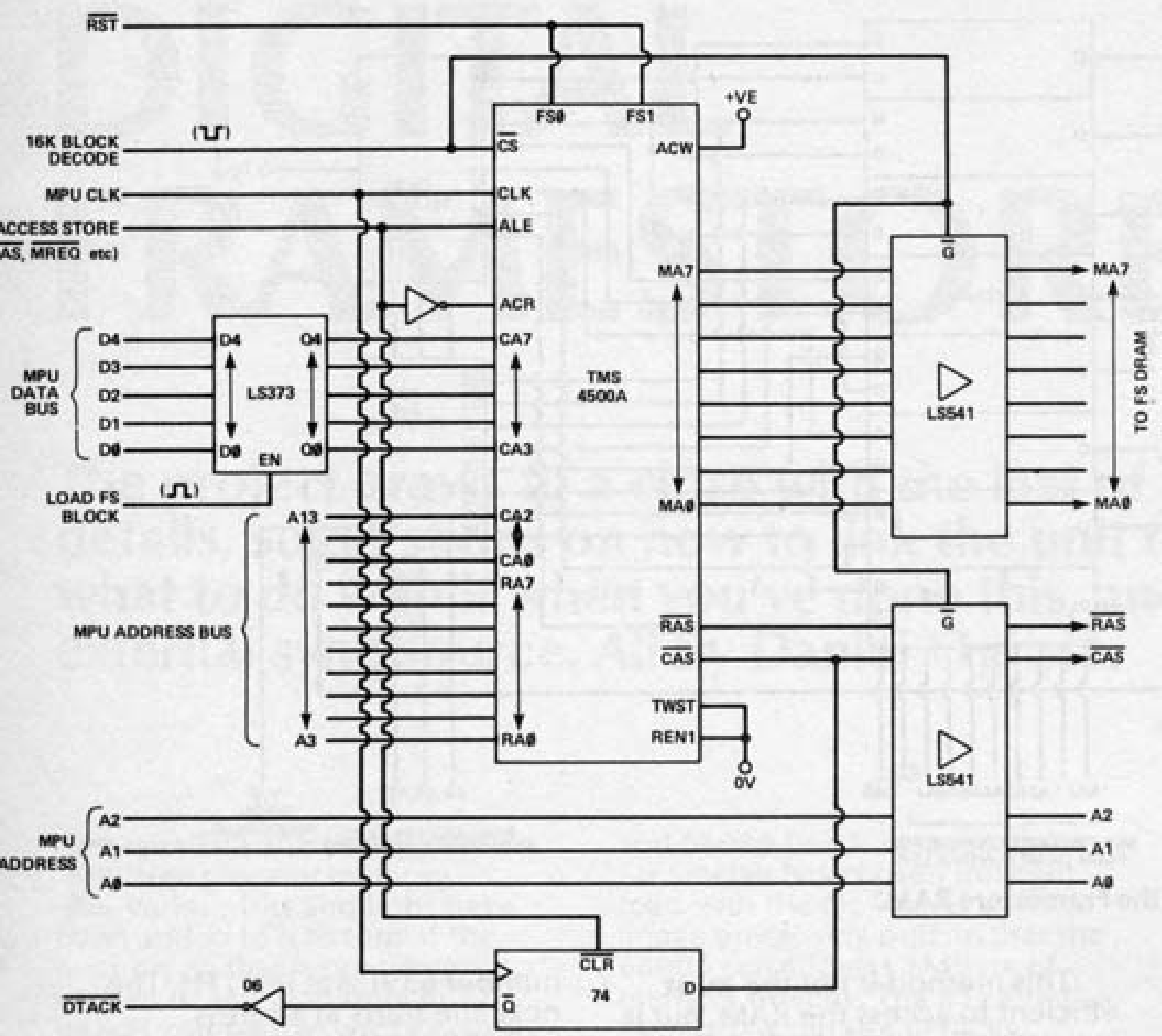
Although slightly more complicated, this method of accessing the DRAMs allows the MPU to "see" a logical memory map. The first pixel stored (top left of field 1) is at address 00000H, the next along the line is at 00001H, etc. The end of the first line (pixel number 639) is at 0027FH. The next line starts at address 640=00280H and ends at 004FFH=640+639. The end of the first field is at (640X-256)−1=27FFFH. The next field starts at 28001H and ends at (640X512)−1=327679=4FFFFH.

In this way, any dynamic RAM controller can access the framestore as if it were a conventional piece of memory. We have also seen that it is necessary to be able to address 327,679 bytes to have access to all of the framestore and this is beyond the addressing

PARTS LIST MEMORY CARD

RESISTORS (all ¼W 5%)	
R1-10	**33R**
R11	**2k2**
CAPACITORS	
Unmarked decoupling — all 100n ceramic	
SEMICONDUCTORS	
IC1, 10	**74LS257N**
IC2-9	**MCM666L20 (64K × 1 200ns DRAM — see text)**
IC11,12	**74LS08N**
IC13	**74LS244**
IC14	**74LS367 (8T97)**
IC15	**74LS138**
IC16	**74LS138N**
IC17,18	**74LS195N**
IC19	**74LS374N**
MISCELLANEOUS	
PCB: wire solder, etc.	

Fig 18. Bank-selected RAM access.



range of most 8-bit microprocessors, which makes it necessary to implement a bank selection technique to enable us to peer through a movable window at the framestore.

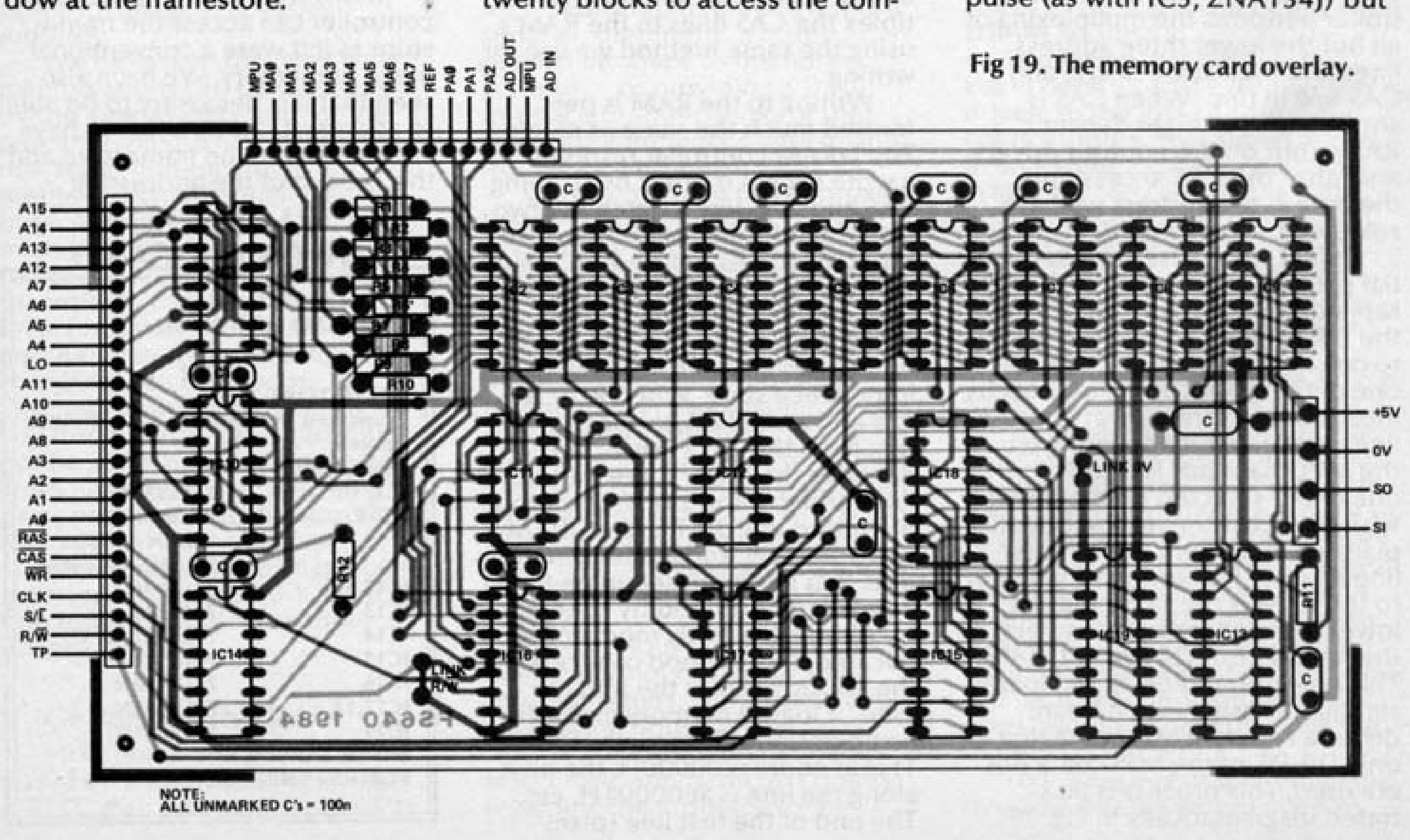
A suggested circuit for this is shown in Fig. 18. If we have, for example, only 16K of memory available in the micro, we will need to be able to select one of twenty blocks to access the complete framestore (20 X 16K = 327,679). The additional upper address lines we require can be stored in a latch by an additional MPU load instruction to select one of the twenty 16K blocks before we perform a memory read or write. Normal read or write operations can now be performed within the block selected.

Synchronising The Framestore

The framestore as it stands is intended to be the master sync generator, ie, it will provide the synchronisation for all the other units in the system connected to it. However, this is not always possible, for example, when using video recorders, off-air broadcasts and some cheap video cameras. The modification described here allows the framestore to be externally synchronised (Fig. 20).

The modification works by replacing IC5 on the control card; IC5 is the sync pulse generator IC. The heart of the replacement circuitry is the TA6993W, which is itself a sync pulse generator, but with the facility to synchronise to an external reference. This IC normally runs off the 500kHz clock input to pin 23 (this should be derived from the 25MHz clock already on the control card). The TA6993W generates an odd field pulse instead of an even field pulse (as with IC5, ZNA134J) but

Fig 19. The memory card overlay.



this is not important, it just shifts our reference point.

The TA6993W contains a phase comparator and a phase-locked loop. When negative going mixed sync pulses are presented on its pin 20 it switches over from the external oscillator to an internal voltage controlled oscillator formed by the RC network on pins 6,22,24. The frequency of this oscillator is varied until the internally generated line and external line input are locked in phase.

The vertical synchronization is achieved by integrating the mixed sync input via the 18k and 1nF capacitor, which generates a field pulse, and using this to reset the vertical line counter. This method produces a quite effective external lock but it will never be as stable as the original method. The trimmer should be adjusted until a stable lock is obtained; be careful to avoid twice line frequency. The switch over between internal and external oscillator is performed automatically and the sync source is indicated by detecting the voltage level on pin 1 and lighting an LED (+6V = external sync).

The front end of the circuit is a sync separator which strips the syncs from the composite video input. The composite video is amplified and clamped by diode D1. This is fed to a comparator formed by the op-amp. The other terminal of the op-amp is fed with a proportion of the signal from the peak detector formed by diode D2 and the capacitor. The comparator threshold is therefore set just up from the sync tips, preventing false triggering due to noise.

To be concluded, finally ... ■

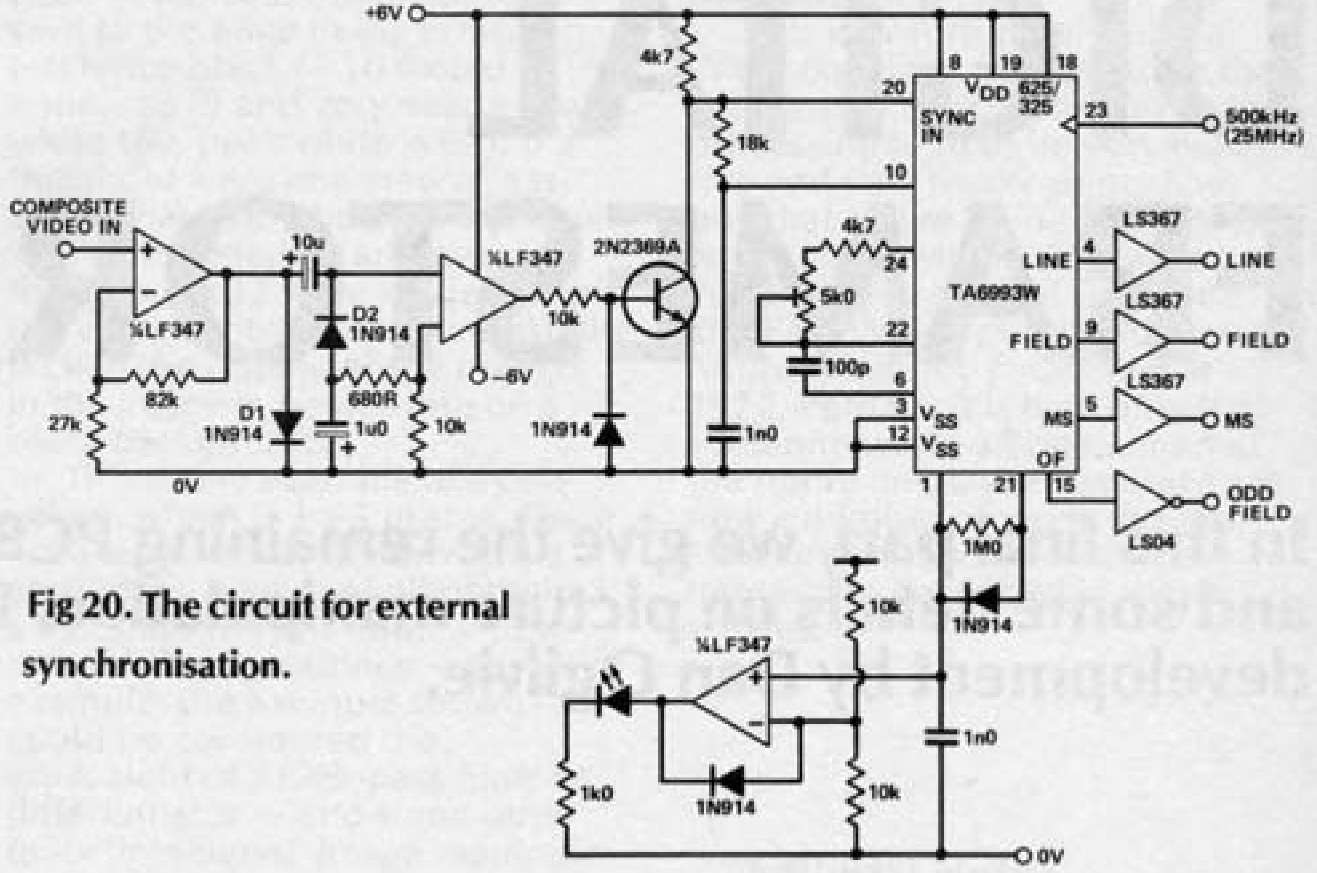


Fig 20. The circuit for external synchronisation.

DIGITAL FRAMESTORE

In this final part, we give the remaining PCB overlay, the Buylines and some details on picture manipulation. Design and development by Dan Ogilvie.

In this section I hope to explain one further method of image enhancement that can be performed on any computer and can produce some very useful results such as edge detection or noise filtering.

Let us for example consider how we might perform a high-pass filter function on our stored image. The simplest way to filter the picture would be an LCR network on the front end before we stored it. This is simple in theory, but variable filters at 6MHz are difficult to design. If we wished to extend the filtering to include low-pass or band-pass filters more switchable elements would have to be added and the design would become increasingly difficult and complex.

Another approach some readers may have encountered would be to take the Fourier transform of the picture, and then to choose just the required coefficients before applying the inverse transform to restore the filtered version of the image. 'Taking a Fourier transform' is just the mathematical equivalent of splitting the signal into its consituent frequencies, the 'coefficients' referred to being the sizes of the different frequency components. Once a signal is split into its component parts this way, we can perform any filtering function we like, no matter how complex. It does not matter that the function we might choose does not have an 'analogue' (LCR) counterpart. Two examples will illustrate how powerful this method can be.

The transmission of television pictures entails a high bandwidth because of the amount of information that has to be sent. This applies also to the storage of information where the large amount of memory (in this case about 393K bytes for one image) means optical discs offer the Hobsons choice.

Any technique to reduce the amount of memory required to represent an image is worthwhile. Some simple methods can be used effective on documents or similar (one bit data) or on some images with little detail. But for any image with detail these techniques are of little use. However by examining the Fourier coefficients and running complex selection routines to select only those necessary to retain picture detail we can reduce the memory requirements dramatically. There are stores of a complex 24 bit colour (8 bits of red, green and blue) being reduced to one bit with little observed degredation of the image! Further reductions are possible if a degree of degredation is considered acceptable.

One further example is the removal of motion blur from a picture — for example, a photograph of a race horse passing by, taken with a stationary camera. Simplistically, if we could examine the Fourier coefficients we should be able to identify the coefficients associated with the race horse's velocity and remove it from the spectrum, leaving the image without the blur.

Fourier analysis certainly offers

BUYLINES

No particular problems should be experienced with any of the components in the framestore. As was stated, the DRAMs used were Motorola MCM6664L20. In view of the number used it is well worthwhile shopping around for the best price. STC should offer a competitive price and accept Access and Barclaycard (tel. 0279 26777).

The ADC/DAC are obtainable from MCP Electronics Ltd, Alperton, Wembley, Middlesex HA0 4PE, tel. 01-902 5941. The full type numbers are ADC-TDC 1014CJ7 and DAC TDC 1016J5C8. They have a minimum order of £25, which will not be a problem in this case!

The crystal is a special and was obtained from IQD Ltd, North Street, Crewkerne, Somerset TA18 7AR, tel. 0460 74433.

The TTL is standard. The 74F devices can be obtained from Hi-Tech Components, Gilray Road, Diss, Norfolk (tel. 0379 4131), or STC amongst others.

The PCBs are available from our PCB Service; six memory and one each of the control and ADC/DAC cards are required.

The TA6993W used in the external sync circuit will have to be shopped around for. It is an RCA device. Try Macro Marketing (06286 4422) or STC.

us a lot in terms of the range of picture enhancements, but unless you have indeterminable patience or a PDP11 to hand, I do not think these techniques can be applied; I would certainly be interested in hearing from anyone who is attempting it.

Let us look at a technique that can be performed on images and is within the scope of our home micro.

The techniques discussed above would be performed in the frequency domain, ie, we would be manipulating coefficients of terms that represented all the frequencies present in the image. We could however manipulate the image in the spatial domain. In other words, by looking at the adjacent pixels to the one we are operating on and only changing it dependent on its own and the local pixel value we can perform a number of techniques similar to those we saw at first in the frequency domain.

The technique itself is called convolution. It is the spatial equivalent of Fourier analysis. Mathematically, it is sometimes possible to convert a Fourier operator to a convolver but generally speaking convolution is not so flexible.

Convolution takes each point on the screen, and from this and the immediately adjacent pixels, it generates a new pixel value. How this works and what it is capable of is probably best demonstrated by an example.

Fig. 19a shows a section of an image which is boring, absolutely flat and dark grey in tone. Fig. 19b shows a convolver. How it is used is as follows: the central term in the convolver ('9' in this case) is placed over the pixel to be operated on, and the pixel and the surrounding pixels are multiplied by the equivalent terms in the convolver, as shown in Fig. 19c. All the resultant terms are added up to form the new pixel value, as shown in Fig. 19d. In this case the new pixel value is exactly the same as the old value. (Note to the mathematically-minded — this is not true matrix multiplication, thank Heaven!)

Let us suppose that a little further to the right of the same image there is an increase in the brightness, spreading over a couple of pixels, Fig. 19e. Applying the same convolver gives the values in Fig. 19f. Notice the extent to which the edge has been enhanced, the pixels adjacent to the edge being turned to reference black (−10 would appear as 0) and very nearly peak white (60, peak white is 63). If a threshold were imposed of, say, 32, so that every below 32 in pixel value were made 0 and every thing above 32 were made 63, this convolver would have successfully picked the outline of the objects in the image in peak white on a black background.

This is one example of a convolver, which is 3 x 3 in size. There are a large variety of convolvers around, by no means all of them 3 x 3 in size. They perform a number of filtering routines — for example, the example shown could be considered the equivalent of a high-pass filter or differentiator — and some other quite 'intelligent' image manipulation functions.

(a)

LINE No. \ PIXEL No.	23	24	25	26	28	29	30
110	10	10	10	10	10	10	10
111	10	10	10	10	10	10	10
112	10	10	10	10	10	10	10
113	10	10	10	10	10	10	10
115	10	10	10	10	10	10	10

(b)

$$\begin{vmatrix} -1 & -1 & -1 \\ -1 & +9 & -1 \\ -1 & -1 & -1 \end{vmatrix}$$

(c)

$$\begin{vmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{vmatrix} \bullet \begin{vmatrix} -1 & -1 & -1 \\ -1 & +9 & -1 \\ -1 & -1 & -1 \end{vmatrix} = \begin{vmatrix} -10 & -10 & -10 \\ -10 & +90 & -10 \\ -10 & -10 & -10 \end{vmatrix}$$

(d)

$$8 \times (-10) + 90 = 10$$

(e)

LINE No. \ PIXEL No.	29	30	31	32	33	34	35
110	10	10	10	20	30	30	30
111	10	10	10	20	30	30	30
112	10	10	10	20	30	30	30
113	10	10	10	20	30	30	30
114	10	10	10	20	30	30	30
115	10	10	10	20	30	30	30

(f)

LINE No. \ PIXEL No.	29	30	31	32	33	34	35
110	10	10	−20	+20	60	30	30
111	10	10	−20	+20	60	30	30
112	10	10	−20	+20	60	30	30
113	10	10	−20	+20	60	30	30
114	10	10	−20	+20	60	30	30
115	10	10	−20	+20	60	30	30

Fig. 19 (a) Digitised section of a flat image; (b) a convolver; (c) applying the convolver to one pixel; (d) the result of applying the convolver — back to square one; (e) a section of image with a change of brightness; (f) the convolver applied to this section.

This is unfortunately all the detail on image manipulation that I can justify here. Books on image processing tend to be very expensive and very heavy going. The one that I have found the easiest to get along with is 'Digital Image Processing' by Rafael Gonzalez and Paul Wintz, published by Addison Wesley Publishing in 1977. Perhaps it is high time that someone wrote a book aimed at the home micro user, as there are now a number of vision systems knocking around besides the framestore described in these articles.

Editorial Note

Far be it for us to discourage anyone from building this project, but a few words on this framestore are needed. From the start, this project was conceived as being for experimenters, and by this, we mean someone with enough experience of electronics to fully understand a system of this complexity.

We hope that all our readers will have gained some insight into the techniques involved in storing and manipulating a TV picture, but, as this framestore uses an awful lot of expensive devices, some of them operating at very high speeds, we would expect relatively few readers to build it.

We strongly advise potential builders to be certain of their ability to de-bug a large digital system such as this before they begin. If you're not certain of your ability — or if you don't have access to the necessary test gear (say at least a 20MHz dual-beam 'scope) — leave it alone until you are fully ready to have a go.

However if you try and fail to build a working Framestore, the author is providing a fault-finding service for just £20 plus the cost of parts for any one board, provided the construction is on a PTH board. He can be contacted on 0245 466936.

We're sorry if this all sounds so negative. However, it's distressing for us — as it must be for the readers concerned — to find a batch of enquiries from people who'll never get their piece of dream hardware to work because they are well out of their technical depth.

PARTS LIST — CONTROL CARD

RESISTORS		C3	68p s.mica	IC4	74F04PC	IC26, 27	74LS138N
R1	560R	C4	33p s.mica	IC5	ZNA134J	Q1, 3, 5, 7	2N2369A
R2, 3, 14, 15	4k7	C5	100n ceramic	IC6	74F163	Q2, 4, 6	2N3906
R4	470R	unmarked	all 100n ceramic	IC7	74LS138		
R5, 9	10k	capacitors		IC8	74LS08		
R6, 8, 10-13	3k3			IC9	74LS20N		
R7, 16, 19	1k0			IC10, 17	74LS74N	**MISCELLANEOUS**	
R17, 20	220R			IC11-14	74LS151N	L1	single turn 18SWG
R18, 21	22R	**SEMICONDUCTORS**		IC15	74LS193		enamelled wire
R22-26	2k7	IC1, 28	74F74PC	IC16	74LS32N	X1	25.625 MHz
CAPACITORS		IC2	74LS04N	IC18	74LS00N		crystal
C1, 2	100p s.mica	IC3	74LS374N	IC19-25	74LS193N	PCB, wire, etc.	

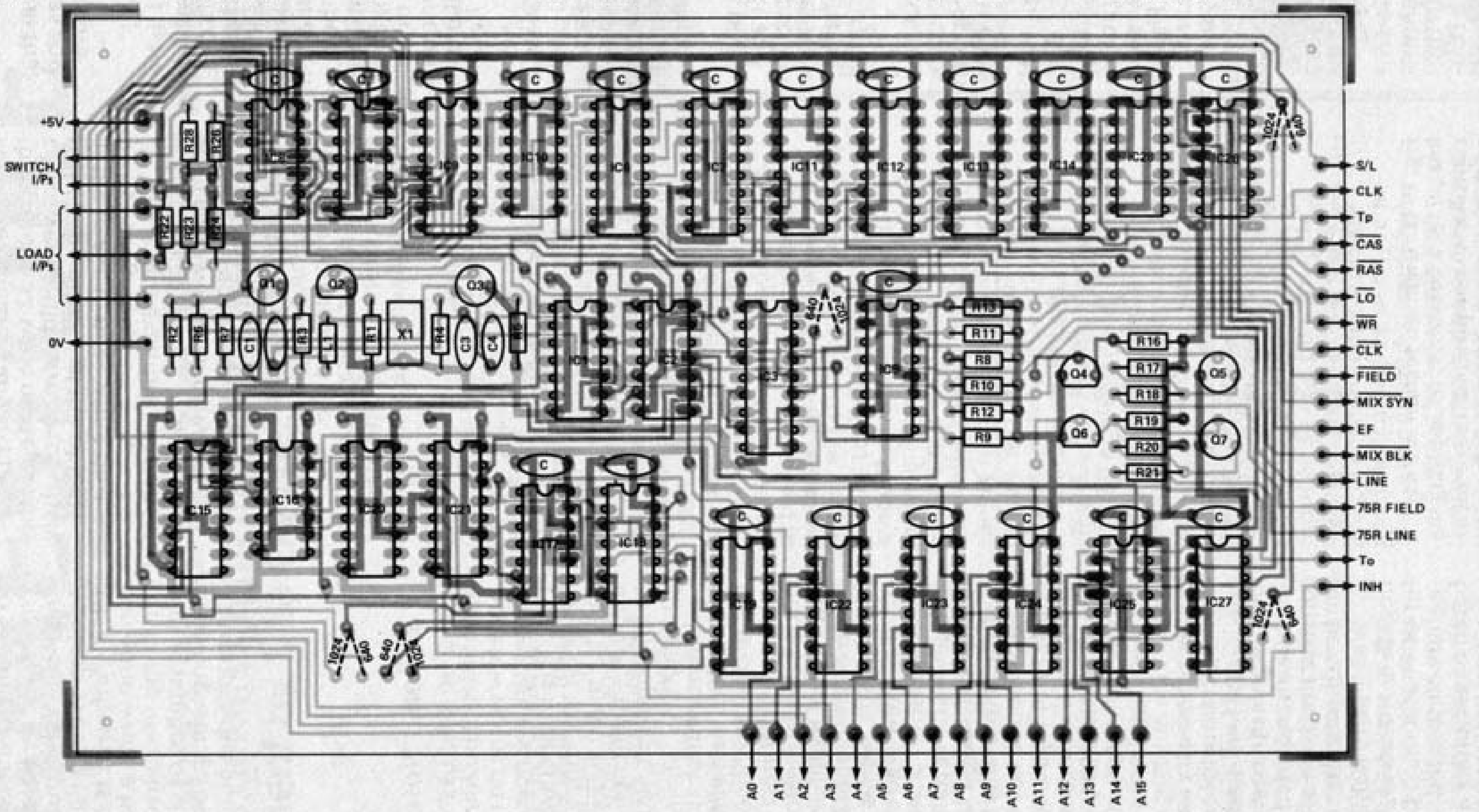


Fig. 20 The final overlay, of the control card. Just a few things have changed between this overlay and the original circuit diagrams published back in Part 1. Firstly, the downward-pointing arrow close to pin 8 IC1 should have been labelled $\overline{\text{CLK}}$. The resistors around the inputs to IC8a and b and IC9a should be labelled R22 to 26. Finally, there were two devices marked IC17a: the one next to IC7 has here become IC28. However there is no reason why the unused half of IC1 should not be used for this purpose, if an alternative PCB lay-out were used.