



GRAPHIC TV DISPLAY

D. M. de Boer

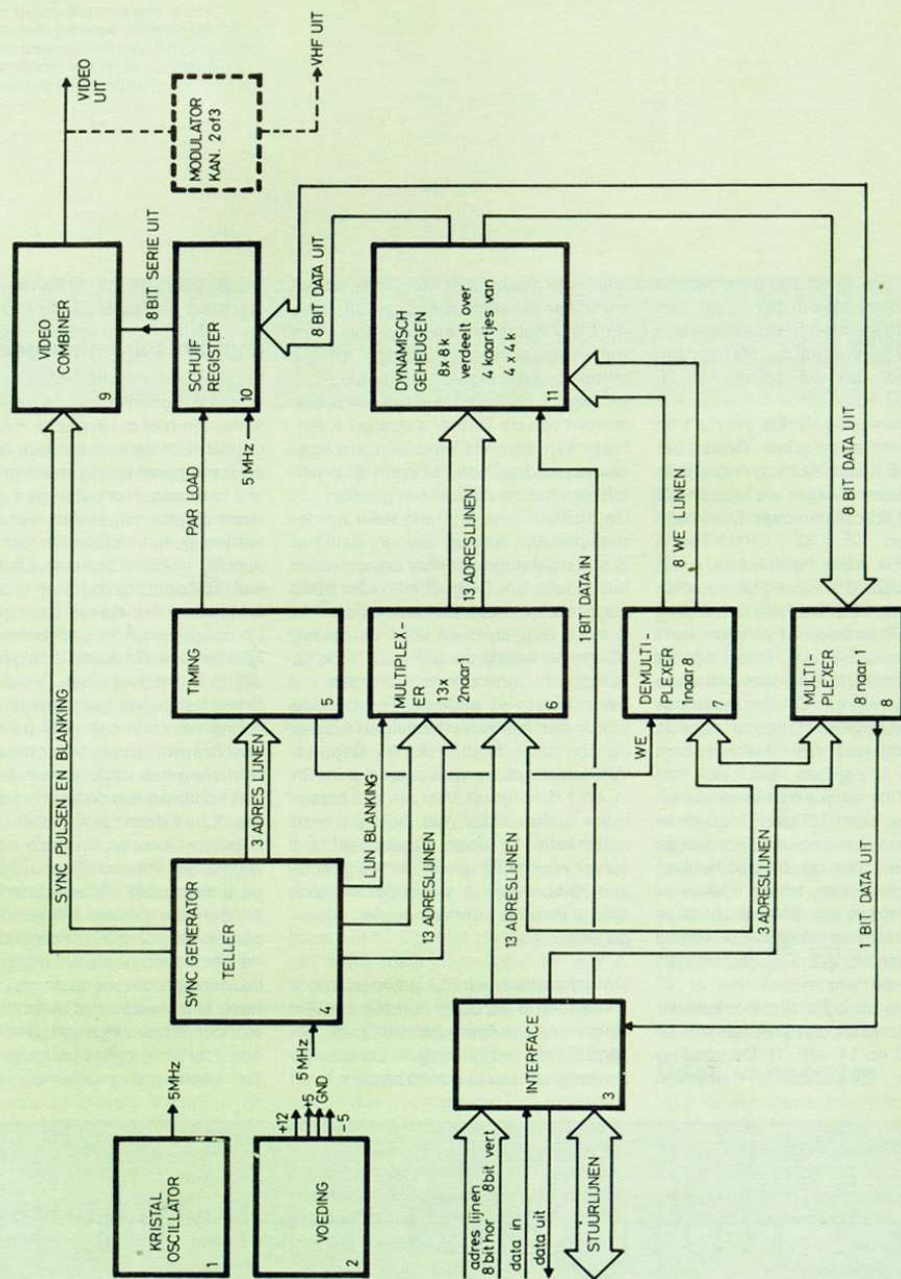
This TV display is the first link. At the input you will find 8 address lines and a few control signals. With these connections it is possible to switch any dot (out of a total of 65536) on the screen on or off. Naturally, control by a microcomputer is the most obvious choice. At a later stage this display will get its own microprocessor, turning it into a graphic/alphanumeric terminal.

A smart design

This design has been made so that you can build either a cheaper or a more expensive version. The cheaper version works with a quarter of the normal amount of memory. The consequence is a coarser grid (128×128 dots). The result is not bad, by the way; the TV on the cover photo also worked with a 128×128 grid. Besides black and white, the more expensive version can also display two shades of grey. To achieve this the memory capacity must be doubled. With this design it is always possible to upgrade to a more expensive version later on. Extensions such as a colour version or the addition of a light pen (real drawing on the screen!) are among the possibilities and are currently being investigated.

Design philosophy

In the design we started from the premise that the whole thing should be as flexible as possible. The first choice to be made is: are we going to build a graphic display or an alphanumeric display? An alphanumeric display can only show letters and digits. A graphic display, on the other hand, can also display lines of any length. By combining lines, every possible figure can be displayed, so all letters too! It will be clear that the graphic display is by far preferable. Apart from the ability to display graphs, figures, etc., the graphic display also makes it possible, for example, to give the letter I a little less space and the letter M a little more. In an alphanumeric display all character widths are matched to the widest letter. Furthermore, the graphic display is able to show larger letters and moving figures. The graphic display has one drawback: it needs much more memory. Recently, however, memory chips have dropped sharply in price (4K dynamic costs about f 15) so that the extra cost is outweighed by the many extra possibilities. The choice therefore fell on the graphic display. In choosing the components, the criterion was: which option is the cheapest? Naturally care was taken that the cheapest solution was also a good solution.



The block diagram (fig. 1)

Fig. 1 shows the block diagram of the complete circuit. The TV display works with a grid of 256 dots horizontally and 256 dots vertically. In total this gives 65536 dots, so in principle we need a memory of $64\text{ K} \times 1\text{ bit}$ ($1\text{ K} = 1024$). A maximum of 256 dots can be written on one line. This must happen within $51.2\text{ }\mu\text{s}$ (the sync lasts $12.8\text{ }\mu\text{s}$), so each dot is written in 200 ns . The 64 K one-bit memory would therefore have to deliver new data every 200 ns . This is (still?) too fast for most memory chips, so we have to use a shift register; in this case an 8-bit shift register (74166). The 64 K memory is therefore organised as $8 \times 8\text{ K}$. The memory now has to read out 8 bits simultaneously once every $1.6\text{ }\mu\text{s}$ (8×0.2). The shift register makes sure these 8 bits end up on the screen, while the memory has ample time to prepare the next 8 bits of data. The sync generator (block 4, fig. 1) not only provides the necessary sync pulses but also generates the addresses. The counter starts at the top left of the TV screen at 0000 (HEX). At the end of the first line the counter stands at 00FF. During the sync pulse the counter is used for the timing of internal signals. During this time the multiplexer (block 6) switches over the address lines, so that the computer is connected to the memory and data can be written or read. At the start of the next line the counter neatly stands at 0100 (HEX). The first byte therefore represents the vertical address. The lines are numbered from 00 (top) to FF (bottom). The second byte represents the horizontal address. The 256 dots on a line are numbered from 00 (left) to FF (right).

In this way every dot on the picture has its own address. Because the memory reads out 8 adjacent bits at once, only 5 address lines remain horizontally. (5 address lines give $2^5 = 32$ different addresses. Each address has 8 bits, so in total there are again $8 \times 32 = 256$ different dots.) The whole circuit works with 8 vertical address lines and 5 horizontal address lines, so 13 address lines in total. The 3 unused address lines, which are generated in the sync generator, take care of the timing in the rest of the circuit. These three address lines run through all 8 possible addresses within one cycle of $1.6\text{ }\mu\text{s}$. By feeding these addresses to a binary/BCD decoder (timing, block 5), we have divided this $1.6\text{ }\mu\text{s}$ period into 8 time slots of 200 ns . From this BCD decoder we derive the timing for signals such as $/\text{WE}$, $/\text{CE}$, parallel load (shift register), etc.

We have now briefly discussed nearly all the blocks that are strictly necessary (4, 5, 6, 10 and 11, fig. 1). The power supply (2) and the crystal oscillator (1) need no explanation here yet. The video combiner (9) combines the sync pulses, blanking pulses and video information into the complete video signal. The nicest solution is to feed this video signal directly into the TV. If necessary a VHF modulator can be added, but this does not benefit picture quality.

Blocks 7 and 8 (multiplexer and demultiplexer) make the $8 \times 8\text{ k}$ memory look like $1 \times 64\text{ k}$ to the computer. This is not strictly necessary, but it requires only 2 extra ICs and offers many advantages. First, the software for the computer becomes simpler because we have only 1 data bit. Second, the number of connections between computer and display is reduced: we now have only 1 data line in and 1 data line out. There are 3 extra

address lines, but we still save no fewer than 11 lines. This number grows even larger if the memory is doubled so that grey shades can be displayed.

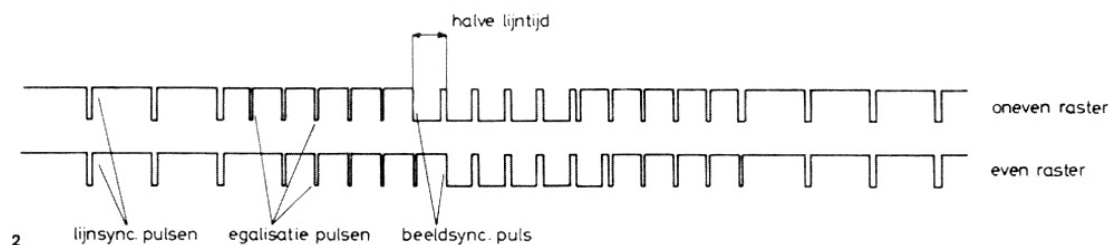
So the computer simply has 8 horizontal and 8 vertical address lines. Of the horizontal lines, as mentioned, 5 go to the memory. The remaining 3 are used to write to only 1 of the 8 locations, or to read only 1 of the 8 bits.

Dynamic memory

Because dynamic RAMs are simply much cheaper than static ones, dynamic RAMs are used here. The drawback that these cells have to be refreshed every 2 ms does not apply here, because the memory is read out continuously. For refreshing it is sufficient to read address lines A0 ... A5. The state of /CS is irrelevant here. Addresses A0 ... A4 are run through during every line. Address A5 belongs to the line addresses. This address is wired so that the entire memory is refreshed within 4 lines of the TV screen. These 4 lines are written in $4 \times 64 \mu\text{s} = 256 \mu\text{s}$, so we need not worry about refreshing.

Writing data into memory can take place during line and/or frame retrace. In this circuit we chose to write only during line retrace. During this retrace we have exactly enough time to write 8 dots. Writing during frame sync makes little sense, because the microprocessor cannot work fast enough to supply more than 8 dots per $64 \mu\text{s}$ anyway. Moreover, writing to memory during frame sync causes problems with refreshing: the total suppression pulse during frame retrace lasts $56 \times 64 \mu\text{s} = 3584 \mu\text{s}$. If we used this time to write to memory, we would have no guarantee at all that all memory locations would be refreshed after 2 ms. All in all, it is much more practical to write only during line retrace.

For reasons of cost we introduced a limitation. The TV display is fast enough to write 8 dots every $64 \mu\text{s}$. The computer must always supply the addresses for these dots. This means we need a buffer memory to hold the 8 vertical address bits and 8 horizontal address bits. By including only one buffer for the 16 address bits, the write speed is limited to 1 dot per $64 \mu\text{s}$. On the one hand this saves a considerable amount of hardware; on the other hand the computer/TV-display combination has not become much slower, since in practice $64 \mu\text{s}$ is very little time to compute a new dot and pass it to the screen. Should a much faster microcomputer ever be used, the circuit can always be adapted.



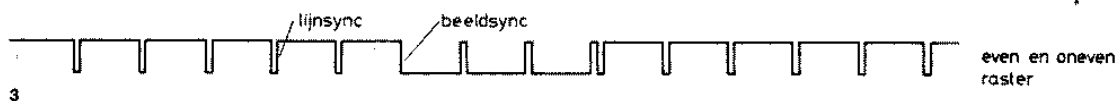
Structure of the TV picture

The TV picture consists of 625 lines, divided over 2 interleaved fields of $312\frac{1}{2}$ lines each.

This interleaving of the fields (interlacing) gives the impression that the picture rate is 50 Hz instead of 25 Hz. For moving pictures this is a fine solution. However, if we have only a single lit line, it appears to jump up and down. That is why in this design both fields are made to fall exactly on top of each other, giving a steady picture. As a result the number of lines on the screen is reduced to 312. Of these, roughly 25 more are lost during frame retrace, so in principle we could use 287 lines. For practical reasons this has been reduced to the 256 mentioned earlier, so that the vertical address occupies exactly 1 byte.

To make the 2 fields fall exactly on top of each other, we must deliberately deviate from the CCIR standard as far as the sync pulses are concerned. Fig. 2 shows the sync pulses according to this standard. Because the frame sync in the even field is shifted by half a line period relative to the frame sync in the odd field, interlacing (the interleaving of the fields) is obtained. In the TV the frame sync is separated from the line sync pulses by integration. To ensure that the integrating capacitor in the receiver has the same initial charge at the start of the frame sync pulse in both the even and odd fields, so-called equalising pulses are added halfway along the line. Put simply, the TV now does not notice that the frame sync pulse sometimes starts halfway through a line. The line sync pulses must continue during frame sync to keep the line oscillator in step. The equalising pulses have no effect on the TV's line oscillator, because this oscillator cannot follow the twice-as-high frequency of the equalising pulses anyway.

In our design we want both fields to coincide exactly. To achieve this, the frame sync of the even and odd fields must start at exactly the same relative moment. This also removes the need for equalising pulses. The sync pulses as we need them are drawn in fig. 3. We saw little point in adding an 'interlace on/off' switch, because proper interlacing only results if the equalising pulses are generated as well. That would again require more components in the sync generator, making the whole circuit more expensive. Given that interlacing is undesirable anyway (all the letters seem to 'dance'), we dropped it entirely in this design.



The sync generator

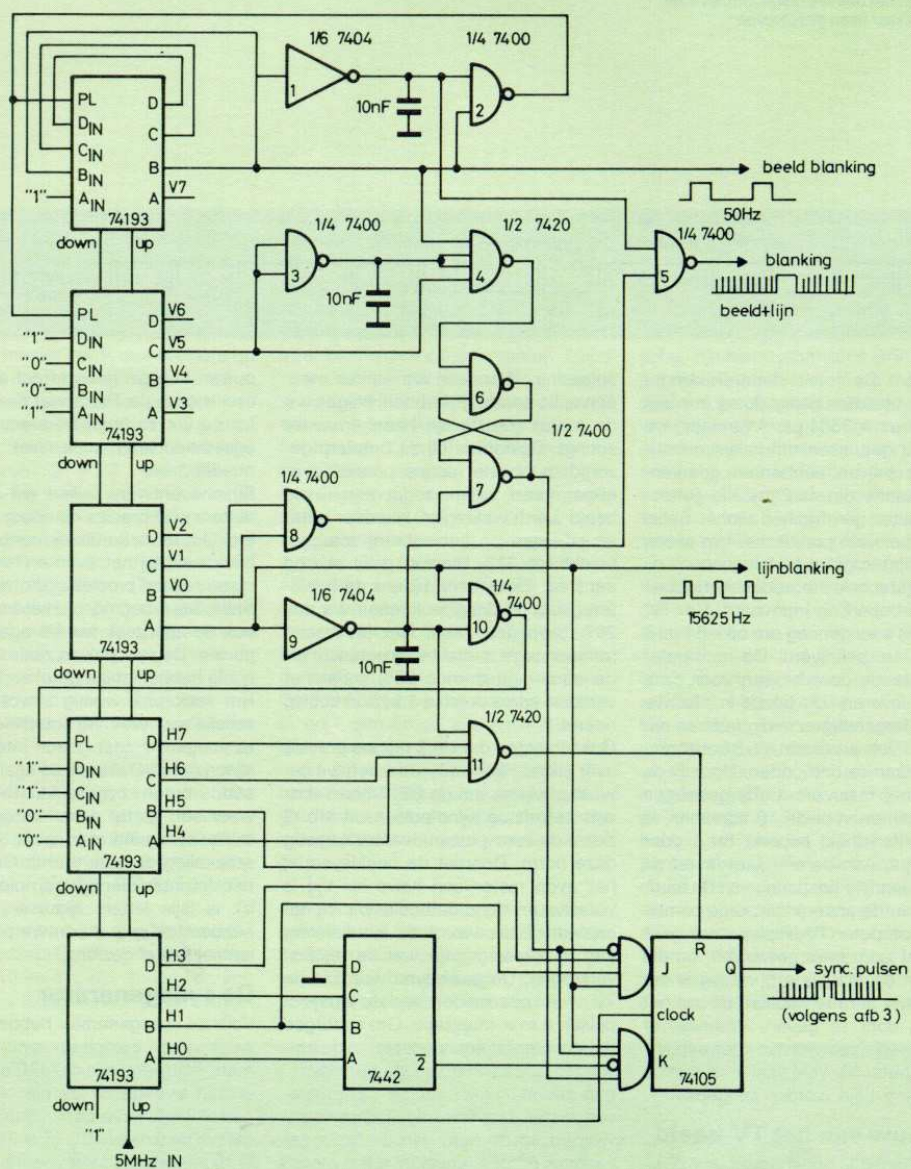
We deliberately did not use a complete sync IC for the sync generator. Firstly these ICs are rather expensive, and secondly they are not always easy to obtain. In addition, we needed the 5 divider ICs (5×74193 , fig. 4) anyway to generate the addresses. By adding a few gates it is now quite simple to build a sync generator.

We start the explanation of its operation at the bottom left of the schematic in fig. 4. Here 2 counters generate the horizontal addresses (H0 ... H7). We tie the 'DOWN' input to a logic '1'. As a result the 'DOWN' input stays at logic '1' throughout the whole divider chain, and we can simply tie all unused inputs in the circuit to this logic '1' level. The

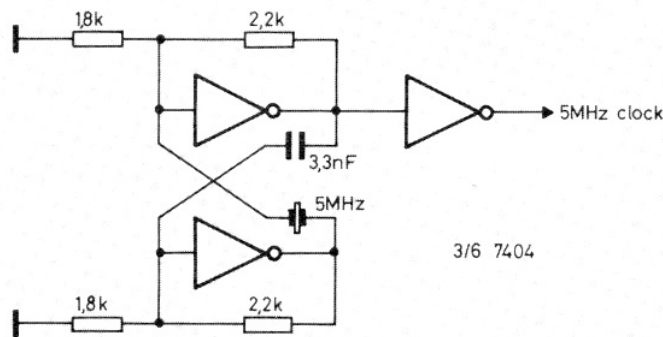
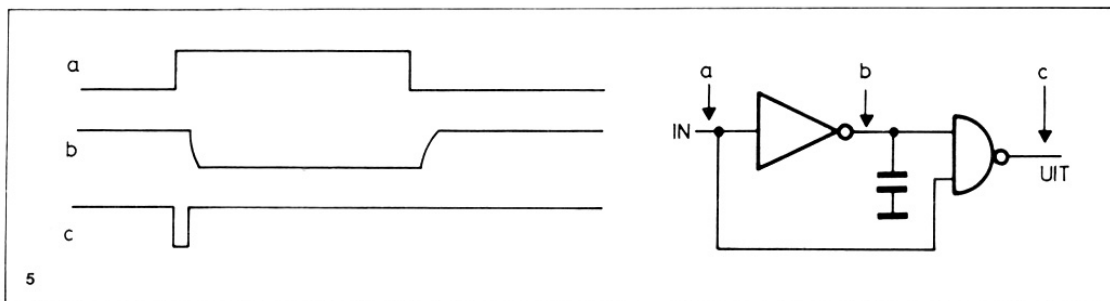
'UP' input is fed with a 5 MHz square wave. When the first 2 counters have completed a full cycle (after 256 clock pulses), $256 \times 200 \text{ ns} = 51.2 \text{ }\mu\text{s}$ will have elapsed. This is exactly the 'visible' time of 1 line on the TV screen. After this the circuit must generate a line sync pulse and a line blanking pulse. The sync pulse makes the TV's line oscillator start a new sawtooth. The blanking pulse ensures that the line is not visible during retrace. The blanking pulse must be $12.8 \text{ }\mu\text{s}$ long, so that the total line time comes to $64 \text{ }\mu\text{s}$. When the first 2 counters have completed a full cycle (after $51.2 \text{ }\mu\text{s}$), output A of the 3rd counter goes from '0' to '1'. The circuit around gates 9 and 10 generates a negative-going pulse on this edge. In fig. 5 this part of the circuit is shown separately for clarity. Signals a and b are fed to the NAND gate. Because signal b is delayed with respect to signal a, signals a and b will both be '1' for a short time. This produces a short negative-going pulse at the output. On a falling edge at the input both signals will be '0' for a short time; the NAND output then simply stays '1'. Back to the schematic of fig. 4. We see that the rising edge of output A of the third counter, via gates 9 and 10, delivers a short negative-going pulse to the 'parallel load' input of the second counter. This loads that counter with the fixed preset value '1100'. It now takes 64 clock pulses before the bottom 2 counters are back at 0. These clock pulses last exactly $64 \times 200 \text{ ns} = 12.8 \text{ }\mu\text{s}$. Output A of the third counter will then have gone from '1' back to '0'. At the same time the line address (V0 ... V7) has been incremented. Output A (3rd counter) thus supplies the blanking pulse for line retrace. The sync pulse is made by the JK flip-flop 74105 (bottom right). Its clock input is triggered continuously, so the state of the Q output is determined by the signals on the J inputs, the K inputs and the reset. The combined JK input of the 74105 is connected to 'line blanking'. As a result, during the visible part of the line the state of the JK flip-flop can only change through 'reset'. So a sync pulse can never occur during the visible line. During line blanking the J and K inputs are enabled. The J inputs receive the condition for 'sync on', the K input the condition for 'sync off'. The J and K inputs are connected to the counters in exactly such a way that a correct sync pulse results. Because one K input is connected to the /J input, the conditions for sync on and sync off can never occur simultaneously. One of the K inputs is connected to the output of gate 7. Normally this output is '1' and has no effect on the 'sync off' condition. We will come back to the purpose of this connection.

So we now know how the line sync pulse and line blanking pulse are produced. After each line (with blanking) the vertical address (V0 ... V7) is incremented. When we have gone through a complete picture (256 lines), output B of the top counter becomes '1'. The '0' to '1' transition of this output again produces a negative-going pulse at the output of gate 2. This loads the top 2 counters with a fixed preset value. Only after 56 lines will the vertical address counter be back at 0. We have then gone through exactly the 312 lines of one picture, and output B of the top counter has returned to '0'. This output is therefore '1' during the 56 lines of frame retrace, and thus forms the frame blanking. Line and frame blanking are combined, inverted, at gate 5, where a total blanking signal is produced.

We now have all the necessary signals except the frame sync. To get the picture roughly centred on the TV, the frame sync must fall about halfway through the frame blanking. The frame sync is created when V5 goes from 0 to 1. The familiar circuit around gates 3 and 4 forms a short negative-going pulse on this transition. Because gate 4 is additionally connected to the frame blanking, this pulse can only occur during frame retrace. So when V5 goes from 0 to 1 during frame retrace, the short pulse at the output of gate 4 sets the flip-flop (gates 6 and 7). As a result the output of gate 7 becomes '0' for 3 lines; gate 8 makes sure the flip-flop (gates 6 and 7) is reset at the right moment. Because the output of gate 7 is '0' for 3 lines, the K input of the 74105 connected to it also stays '0'. The 'sync off' condition cannot then occur, and the sync pulse stays high (frame sync as in fig. 3). Just before the new line sync starts, the 74105 is reset by gate 11. The falling edges of the line sync and of the frame sync are therefore always at the same (relative) moment, because the 'sync on' condition never changes. Only the 'sync off' condition is modified for 3 lines in each field. In this way we immediately obtain the composite sync pulses.



The three capacitors in this sync generator set the correct pulse width at the outputs of gates 2, 4 and 10. These pulse widths are not critical, but they must not be wider than about 3 μ s, otherwise the PL pulse of the 2nd counter blocks the pulse on the 'UP' input. The minimum width is about 20 ns. Thanks to these wide limits, the three capacitors in this circuit can never cause instability. The other pulses are generated from the counter states and are therefore rock-solid. A synchronous counter type was deliberately chosen, because otherwise the long divider chain would never 'flip over' in time. Outputs C and D of the top counter can optionally be used as a field counter. The 7442 at the bottom centre (fig. 4) is actually superfluous for the sync generator, but as the block diagram shows, we use the outputs of this binary/BCD decoder for the timing of signals such as /CE, /WE, parallel load (shift register), etc.

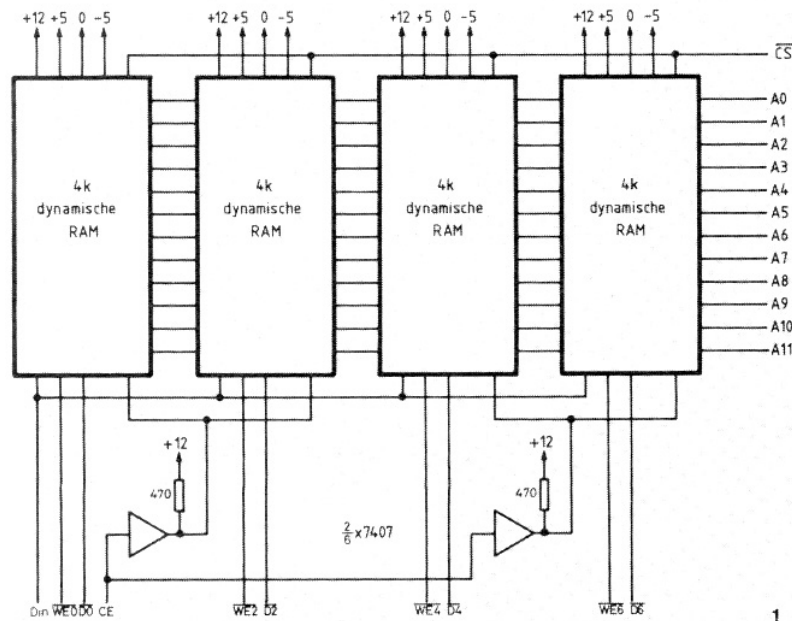


The 5 MHz oscillator

Fig. 6 shows the schematic of the crystal oscillator. The oscillator is formed by 2 inverters. The third inverter acts as a buffer and at the same time ensures that the output signal is a clean square wave. The crystal mainly ensures the stability of the generated frequency. The exact frequency is less important; it must lie between 5 MHz and 5.05 MHz.

The RB TV display could be seen in the flesh at the Firato [Dutch electronics trade fair]. On the whole the public's reaction was very positive. Besides the many nice to very nice drawings (using a number of push buttons visitors could make their own drawings),

quite a few questions came up. Under the heading 'For clarity', elsewhere in this article, the most frequently asked questions are answered. In this 2nd part the theoretical treatment will be practically completed. Only the memory timing has been moved to part 3. Part 3 will also contain the construction description.



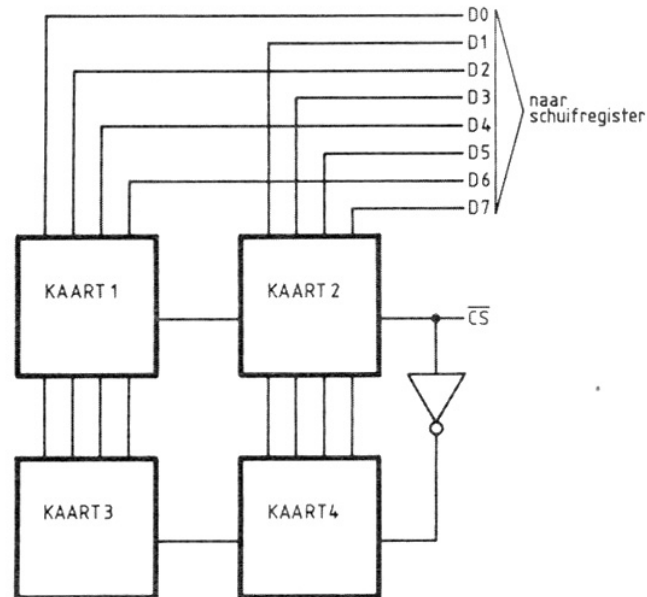
The memory

As announced in the first part, this design uses dynamic 4 k RAMs. Dynamic because dynamic RAMs are much cheaper than their static brothers. The drawback that dynamic RAMs must be refreshed every 2 ms plays no role here, because the memory is read out cyclically. As a result it is automatically refreshed every 256 μ s (see part 1).

Furthermore, this design uses the 22-pin 4 k RAM (industry standard). This too was deliberate, because the 16- and 18-pin versions use multiplexed addresses or combined inputs and outputs respectively. The 22-pin version has all addresses as well as separate input and output available, so control is simplest here.

The memory is divided over 4 cards of 4×4 k each. Each memory card carries four memory chips. The schematic of one memory card is shown in fig. 1. As you can see, all addresses are connected together. This also applies to the /CS connections, so that we can select or deselect the whole card. At the bottom of the schematic you will find the connections D0, D2, D4 and D6. These are the 4 data outputs which, together with four outputs of another memory card, are connected to the 8-bit shift register. On this second card we speak not of D0, D2, D4 and D6, but of D1, D3, D5 and D7. Why we chose this numbering will become clear. The data-in lines are also all tied together. This is possible because we only want to write one bit at a time anyway. By applying the /WE pulse to the right IC, only one of the four bits is written. This also explains why the /WE lines are brought out separately. The clock pulse for the dynamic chips (CE) must be at

MOS level, i.e. 12 V. It is not possible to convert the TTL clock to MOS level once and then distribute it to the four memory cards; this would cause far too much crosstalk on the very high-impedance clock line. Therefore each card gets its own level shifter (7407). Unfortunately, 4 of the 6 buffers now remain unused. However, the IC used is not expensive, so this is no great objection.



2

The cheap version, with 1 memory card

As said, the memory is divided over 4 cards of 4×4 k. This lets us easily reduce the 256×256 dot grid (4 memory cards) to 128×128 (1 memory card). Fig. 2 shows how the 4 memory cards are interconnected.

For clarity, only the data lines and the /CS lines are drawn. The address lines, the Din lines and the CE line are simply looped through. The /WE lines are treated in the same way as the data outputs. (The /WE lines of course do not go to the shift register but to a demultiplexer, fig. 10.) Back to fig. 2. We see that the data outputs of cards 1 and 3 (2 and 4) are connected together. The /CS line ensures that card 1 (2) is selected on the even lines of the TV screen and card 3 (4) on the odd lines. If, to save money, we use only 1 memory card, say only card 2, then the shift register will in any case receive no information during the odd lines, because at that moment cards 3 and 4 would be selected. Their socket is empty, however, so the data lines are 'floating'. Eight logic ones are now clocked into the shift register, which will later be interpreted as 'black'. So all odd lines disappear from the grid, leaving only 128 of the 256 lines.

The same happens horizontally. Because card 1 is absent, bits D0, D2, D4 and D6 will be '1', so that 128 dots on each line always remain black unconditionally. So on the even lines too only 128 dots remain. The result is a grid of 128×128 dots. By 'dot grid' we really mean 'dots' here; a solid white area is not possible. Fig. 3 shows text written in this dot grid, and as you can see this grid is quite acceptable for letters. With a small

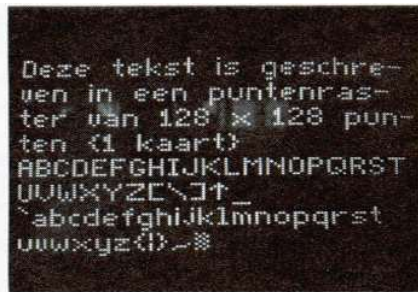
modification we can make the dots larger so that solid white is possible (fig. 4). In time the memory can be expanded to 4 cards, giving a grid of 256×256 dots (figs. 5 and 6).

The addresses

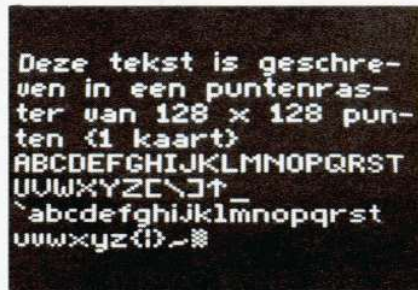
Fig. 7 shows the address control of the memory cards. During the visible part of the TV picture, 'line blanking' is a logic '0'. As a result the multiplexers (3×74157 and 1×7400) pass the addresses generated by the sync generator through to the memory cards. Because the memory produces 8 bits at once, addresses $H_0 \dots H_2$ need not be fed to the memory (see part 1). During the visible part of a line all H addresses are run through; during the visible part of the picture all V addresses are run through. In this way every dot on the screen has its own address.

When we want to change the memory contents, we must be able to feed the address of the bit to be changed to the memory. This must never happen during the visible part of the TV picture, because then the contents of a wrong address would suddenly appear in the middle of the picture. That is why line blanking is connected to the select input of the multiplexers. As a result, during line retrace the address held in the 4 counters (4×74193) is always fed to the memory. This also happens when we do not want to change anything on the screen at all, for the simple reason that it makes the hardware simpler and therefore cheaper. The 4 counters are used here as a buffer between the computer and the TV display. In normal operation the counting function of the 74193 is not used. The only reason counters were chosen is to keep open the possibility of manual operation; we will come back to this. Addresses $h_0 \dots h_2$ are not fed to the memory cards but to the multiplexer and demultiplexer of fig. 10. With these three address lines we determine which of the 8 memory bits must be changed. From the 'outside' of the TV display it now looks as if the display has a memory of $64 \text{ k} \times 1$ bit instead of $8 \text{ k} \times 8$ bit. This makes the software for the display a good deal simpler.

Furthermore, fig. 7 shows that the CS of the memory cards is connected via the multiplexer to V_0 . This means one set of memory cards is selected during the even lines of the TV screen and the other set during the odd lines. This lets us switch easily to 128 lines (cheap version). All this was already explained at the beginning of this part. To limit the number of connections between computer and display, the horizontal and vertical addresses are passed over the same 8 lines. This allows the whole display to be connected to a single PIA (2×8 bits).



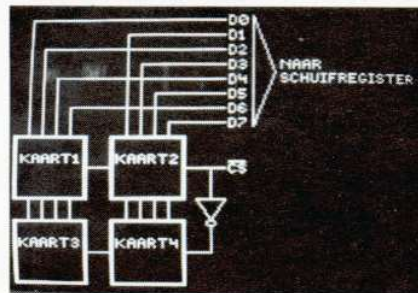
3



4



5



6

Control signals

Of course the computer must provide control signals indicating whether the address presented is a horizontal or a vertical address. In addition it must be possible to make the whole screen white or black in one go (clear). Fig. 8 shows how this is implemented. A BCD-to-decimal decoder (7442, top left) decodes 3 control lines from the computer. Depending on the binary number the computer gives, we have the following functions:

C B A

- 0 0 0 no action
- 0 0 1 put address in X buffer
- 0 1 0 put address in Y buffer
- 0 1 1 change the addressed bit
- 1 0 0 clear the whole display
- 1 0 1 \
- 1 1 0 } reserved for future extensions
- 1 1 1 /

State '0 0 0' is the idle state: no addresses are passed to the display and nothing is written. Output /0 of the decoder is therefore not connected to anything. As soon as the computer outputs '0 0 1', output /1 of the decoder goes low. This output is connected to the clock of the X buffer (fig. 7), so the horizontal address is clocked in. In state '0 1 0' the same happens for the vertical address. Of course the desired address must always be placed on the input address lines hv0 ... hv7 (fig. 7) at the right

moment. Once the correct address is stored in the X and Y buffers, we can give an 'action' command by putting '0 1 1' on the control lines (fig. 8). The result of this command must be that one /WE pulse is generated during line blanking. Fig. 8 shows the circuit that takes care of this.

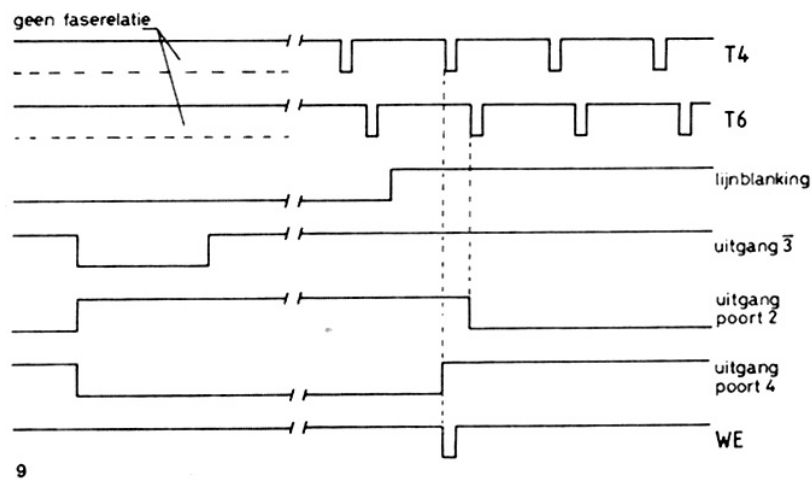
The falling edge on output /3 of the upper 7442 sets 2 flip-flops. The first flip-flop, made with gates 1 and 2, makes the output of gate 2 go to '1'. As soon as line blanking goes high (only then is the address in the X and Y buffers switched to the memory), pulse T4 appears at the output of gate 5. This pulse is used as the /WE pulse. When the /WE pulse appears, the flip-flop around gates 3 and 4 is reset again. As a result pulse T6, via gate 6, resets the first flip-flop (gates 1 and 2). The timing of these signals is drawn

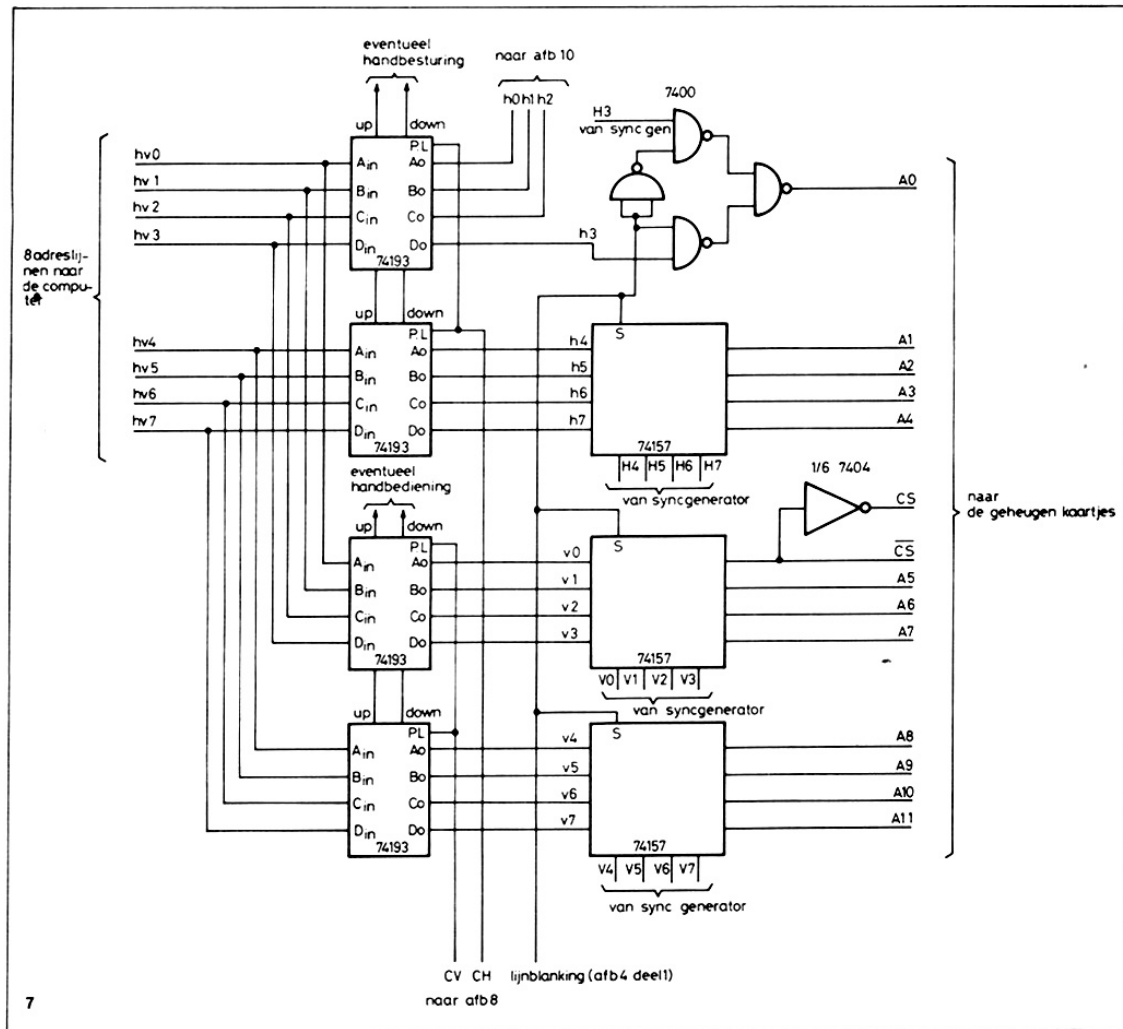
clearly once more in fig. 9.

Changing an arbitrary dot on the screen thus requires 3 operations (clock X, clock Y and 'action'). To keep control simple we deliberately chose the consecutive codes 0 0 0 ... 0 1 1 for this, so that we can always move from one state to the next with a single 'increment' instruction.

In practice it sometimes happens that when the control code changes from '0 0 1' to '0 1 0', state '0 1 1' occurs for a very short time. This would produce a false 'action' pulse.

That is why we feed clock pulse /Ø2 from the computer system to the D input of the decoder. As long as the D input is high, outputs 0 ... 7 are always high as well. The /Ø2 pulse only goes low again after the change from 001 to 010 has taken place. In this way a false pulse can never occur.





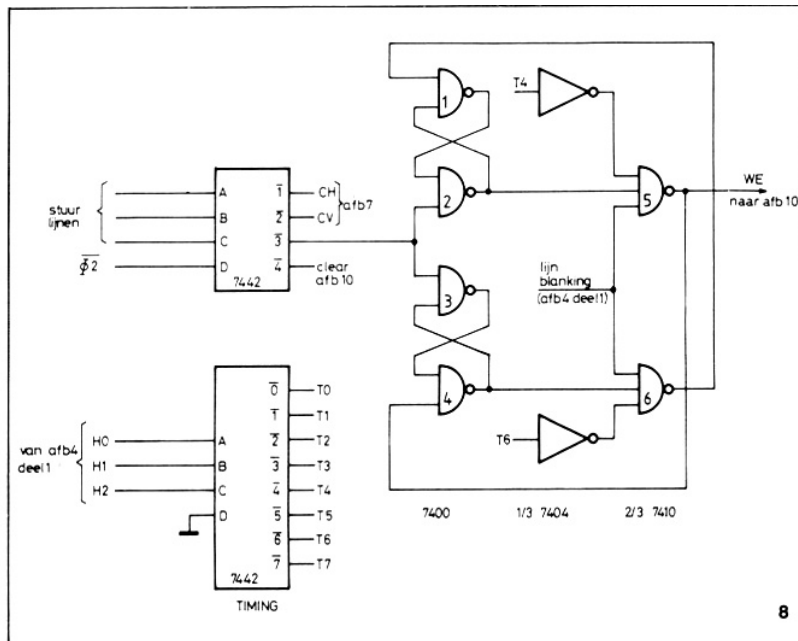
Clear

When we put code '100' on the control lines, output 4 of the BCD decoder goes low. This output is connected to 'clear' in fig. 10. We see that this (via 8 AND gates) pulls all /WE lines to the ICs to '0'. If we make sure this state is maintained for at least 20 ms, the whole display will be filled with a '0' or a '1', making the whole screen white or black. The 20 ms pulse must be produced in software, so we need not include a 20 ms one-shot in the circuit.

The critical reader will have noticed that output 4 of the BCD decoder (fig. 8) can never stay low for 20 ms, because the /Ø2 clock pulse continually drives the D input of this decoder high and low. As long as the clock frequency of the computer system is higher than about 1 MHz and lower than 10 MHz, this system works fine, because there is always a pulse within each memory cycle of 1.6 µs. Most hobby computers run at frequencies within these limits.

The multiplexer (7442) in fig. 10 makes sure the /WE pulse reaches the correct memory IC. In this way we write to only 1 of the 8 addressed bits. (A memory address contains 8

bits.) Which bit is ultimately written is determined by the lowest 3 horizontal address bits (h0 ... h2, fig. 7).



For clarity

This construction description leads to a display whose 65536 dots can be switched on and off individually. With a suitable program, symbols of any width can be placed anywhere on the screen. Character shapes and sizes can be defined by the user. Of course figures can also be drawn. The required programs will certainly be published for the 6502, and probably for the 6800 and 8080. The PCBs used in this design will be included in our PCB service. The design uses ordinary TTL (about f 100). For the memory chips we chose the standard 22-pin 4 k dynamic RAM (4060 from Texas or 2107 from Intel). The cheap version has about f 60 worth of memory, the more expensive version (256 × 256) about f 240. The price of the single-sided PCBs is not yet known. For a total estimated price the power supply must also be included.

In the course of next year we will start construction phase II. The display will then be extended with a keyboard, light pen and a serial/parallel interface. With the keyboard, letters (alphanumeric mode) or coordinates (graphic mode) can be typed in. In graphic mode a straight line is always drawn between 2 points (just as with a plotter). The display can then be used as a combined plotter/teletype.

In the final construction phase we hope to extend this unit into a complete hobby computer.

The data lines

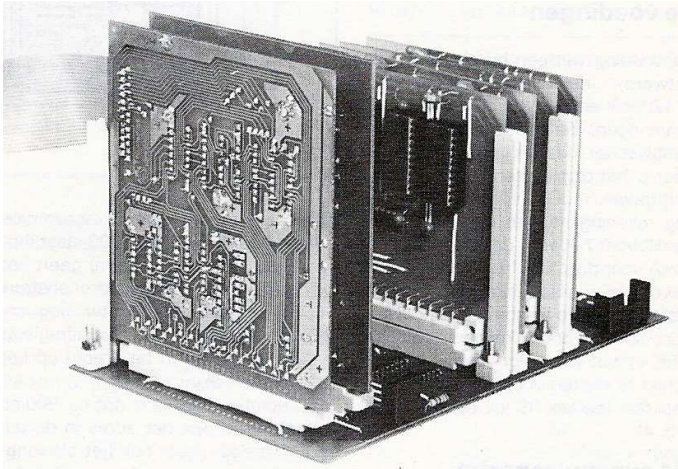
The 8-bit data that the memory delivers once every 1.6 μs is fed to a shift register (74166 in fig. 10). The shift frequency of the shift register is 5 MHz, so a 'bit' appears at the output every 200 ns. The computer must also be able to read back a bit from the screen.

The diagram illustrates a video memory system architecture. At the top, an 8x8 memory array (8x D_{OUT}) provides data to a 74166 shift register. The 74166's H, G, F, E, D, C, B, and A inputs are connected to the memory outputs D0 through D7. Its LOAD is connected to T4 (5MHz), and its CLOCK is connected to the "1" = zwart / "0" = wit signal. The 74166's serial output drives the video signal. This video signal, along with sync and blanking signals from the 3/4 7426 decoder, is connected to a +12V supply to produce composite video. The 74151 8-to-1 multiplexer selects between the 8x D_{OUT} inputs based on address lines h0, h1, and h2. Its A, B, and C outputs are connected to the 7442 3-to-8 decoder. The 7442's outputs (7, 6, 5, 4, 3, 2, 1, 0) are connected to the 7408 2x7408 decoder, which produces 8x WE (WE0 through WE7) signals for the memory array. The 74153 2-to-4 decoder, controlled by STUUR LIJNEN (0 and 1), provides D_{OUT} signals to the memory array and via FF to the computer. The 74151's D0 through D7 inputs are also connected to the 74153's 1C0 through 1C3 inputs.

```
0 0: write the same data
0 1: write inverted data
1 0: write '0'
1 1: write '1'
```

We see that this adds 2 more control lines for the computer. The original intention was to read data back to the computer. In that case the /WE pulse should really not go to the memory (nothing may be written, after all) but to an output flip-flop, which must then remember the state of the bit read. Again to keep the hardware as simple as possible, we always feed the /WE pulse both to the memory and to the output flip-flop. When we want to read rather than write, we must put '00' on the above-mentioned control lines. The old data is then written back into memory, so the contents do not change (despite the /WE pulse). At a later moment the computer can then read the state of the flip-flop. So with the 3 control lines mentioned earlier we determine where the address goes (X or Y buffer) and when the display must write.

With the 2 control lines above we determine whether the addressed dot becomes white, black or inverted, or whether the addressed dot is read from the screen.



Construction

The complete display consists of a motherboard into which an interface board, a sync board and one, two or four memory cards can be plugged by means of edge connectors. Each board is held upright by a card guide. In the photo (fig. 1) we are looking at the copper side of the

interface board. Just behind this board the sync board can be seen. The four memory cards are mounted with their components facing forward. To keep everything as cheap as possible, all boards are single-sided. As a result wire links on the boards could not be avoided, but given the savings this yields, that is hardly an objection. Moreover, wire links have the advantage that extensions and modifications are often easier to make, so the construction remains flexible. The board connectors (DIN 41617) can, if desired, be replaced by fixed wire connections; bear in mind, however, that repairs, extensions etc. will then be more difficult.

A TV with a modulator, or a TV as a monitor?

The first step in building the graphic TV display is making the TV suitable for video signals. By far the simplest way is to use a modulator, so that the video signal is modulated onto a carrier. Nothing then needs to be changed in the TV. The editorial staff tested a number of commercially available modulators. Very reasonable results were obtained with the TV-1 (kit) from UHF Associates, supplied in the Netherlands by Ingenieursbureau Koopmans in Papendrecht. However, none of the modulators met the required standards regarding white-level modulation depth (10%), harmonic suppression, etc. It goes without saying, therefore, that the best result is obtained with a 'real' monitor input. The difficulty here is that no general rule can really be given for converting a TV into a monitor. This modification is therefore strongly discouraged if you are not sufficiently familiar with how a TV works. The only guideline we can give is that the monitor input must come after the last demodulator, i.e. at the point where a video signal first appears in the TV.

With valve (tube) sets the modification is easiest. Here it is usually sufficient to disconnect the control grid of the video output valve from the preceding electronics and to attach the monitor input at that point. With transistor sets we often also have to restore DC levels. If you use your living-room TV, it is advisable to build in a switch to choose between TV and monitor.

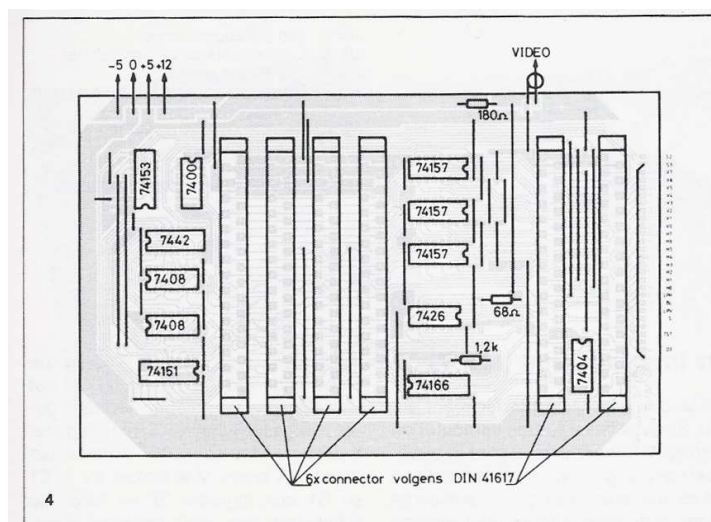
It is often possible to buy working valve TVs for f 10 to f 50, so that you can cut and solder to your heart's content without the risk of quarrelling with your wife and/or

children because they cannot watch TV in the evening.

Warning: Most TVs have no mains transformer, so the chassis may be live. Reversing the mains plug is the cheapest solution; an isolation transformer is by far the safest. It goes without saying that we recommend the latter. The video signal has been made fairly strong: 12 V peak-to-peak. If this voltage is too high (transistor TVs), a small voltage divider must provide the correct output voltage.

The power supplies

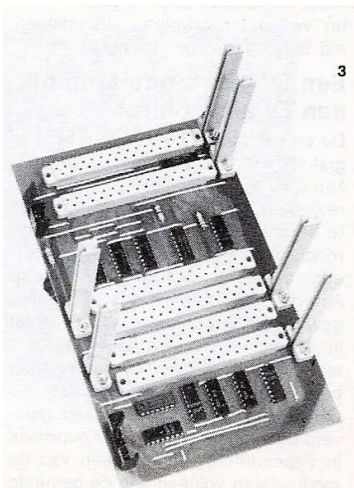
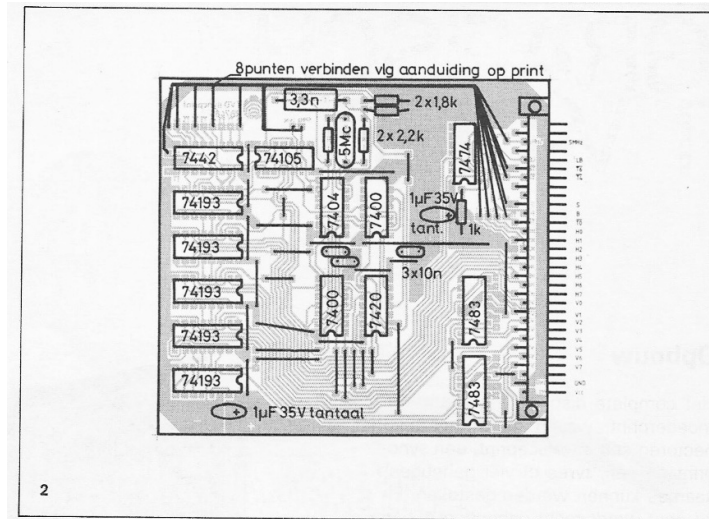
The power supply falls somewhat outside this design; the supply voltages (+12, +5 and -5 V) are the 'usual' voltages available to almost every computer hobbyist. If you want to give the display its own supply, allow for 0.5 A at 12 V, 0.7 A at 5 V and only 1 mA at -5 V for this circuit. In the future the display will be extended with its own microprocessor system. If you want to follow these developments, it is advisable to make the +12 V supply capable of 1 A and the +5 V of 2 A (see also RB July 1978, p. 242, fig. 4).



The sync pulse board

Once your TV is ready to accept video (whether or not via a modulator), we can start assembling the sync pulse board. This board carries the circuit published in September '78, p. 328. It generates the necessary sync and blanking pulses for the TV and also supplies the addresses for the memory cards. There is provision on the board for two extra 4-bit full adders. These make a hardware scroll-up possible (the vertical addresses are increased by an 8-bit number, so the whole picture 'rolls' vertically). For the time being we can simply loop the address lines through, or (if desired) fit the 2 ICs but leave them unused. In the future the built-in microprocessor will handle this scroll-up. The blanking signal, which suppresses the TV's electron beam during retrace, was not quite correctly timed in our prototype. At the moment the horizontal address counter is at 0000 0000 (far left of the screen), the blanking signal releases the electron beam. However, it then takes about another 800 ns before the shift register is clocked and bit 0 actually appears on the screen.

So the 'visible' address is shifted by about 800 ns relative to the address in the counters. Hence the blanking signal must also be shifted. For this reason a flip-flop was added to this final version, which clocks the blanking signal in on T4. This gives the required shift. Fig. 2 shows the component layout of the sync board. Around IC 7442 (top left) you will find the connections T0 ... T7. Normally we only use T0 (CE), T4 (WE) and T6. However, when working with slower or faster RAMs, the timing can be shifted very easily.



The motherboard (fig. 3)

Before the sync board can be tested, we must first assemble the motherboard. To save costs, the holes for the card guides were drilled with the same diameter as the connector holes.

Before assembly starts, these holes must first be drilled out with a 3 mm drill. Then fit all wire links, then the ICs and finally solder in the connectors (DIN 41617).

Remember to use insulating washers when mounting the upper card guides, to prevent shorts to the supply tracks underneath. Fig. 4 shows the component layout of the motherboard. When you have finished the motherboard

and sync board, the first test phase begins. Connect the TV to the display output and connect the supply voltages. Remember that the 12 V video signal must be reduced to about 1 V via a potentiometer of about 1 k Ω . A good value can then be found experimentally later. After switching on, first measure the supply voltages (short circuit on one of the boards?). If the voltages are correct, an even raster with stationary lines should appear on the TV. If in doubt, you can briefly disconnect the video connection and look at the difference. If the raster is not right, the fault is probably on the sync board. In that case check the sync board for shorts, defective or wrong ICs, incorrect

wire links, etc. From the moment the raster is correct, the display can be checked using the display itself as a meter!

Checking the connectors via the TV

The display is arranged so that it can be checked by means of an extra video input. A logic '1' gives a black picture, a logic '0' a white picture. Square waves give a bar pattern, horizontal or vertical depending on the frequency. The extra video input can only handle digital signals, so we must never connect this input to +12 or -5 V. At the far right of the board (fig. 4) are the motherboard connections. One of these connections is marked 'TST', short for 'Test'. We now temporarily attach a test lead to this point. At its end goes a thicker piece of wire that fits into the connector. We start with the connector for the interface board (far right) at pin 31 (top):

PIN	SIGNAL	PICTURE
31	Vcc	black
30	GND	white
29 to 8	inputs	black
7	/T4	thin black vertical stripes
6	/T6	ditto, not in the same place
5	LB	white
4	d4	white
3...1	inputs	black

If a bar pattern appears on the screen for any of pins 29...8 and 1...3, the track from that connector pin is shorted somewhere along its route to an adjacent track. Such a fault can easily occur when soldering the connectors. Once the signals for the interface board are correct, we continue with the connectors for the memory cards. The test must be carried out for each of the 4 memory connectors. Again we start at the top, at pin 6. Pins 1 and 5 must not be tested (+12 and -5 V!!). To avoid mistakes we start at pin 6:

PIN	SIGNAL	PICTURE
6	A11	top white, bottom black
7	A10	two horizontal bars
8	A9	4 horizontal bars
9	A8	8 horizontal bars
10	A7	16 horizontal bars
11	A6	32 horizontal bars (each bar 4 lines thick)
12	A5	bars 2 lines thick
13	/CS	'bars' 1 line thick
14	A4	1 white vertical bar
15	A3	2 white vertical bars
16	A2	4 white vertical bars
17	A1	8 white vertical bars

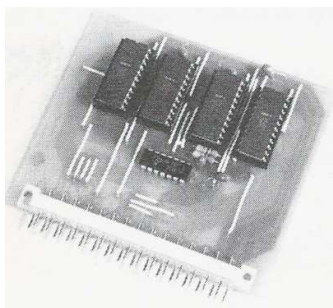
18	A0	16 white vertical bars
19, 20	/WE2, /WE1	black
21	NC	black
22	CE	thin vertical white lines
23, 24	/WE3, /WE4	black
25, 26	/Dout1, /Dout2	black
27	NC	black
28, 29	/Dout4, /Dout3	black
30	DIN	white

The number of bars may not be exactly right because one or more bars fall outside the picture. When all connectors give the correct signals, we must also check the inputs. Disconnect the test lead from 'TST' and connect it to ground. Then, for each memory connector, connect pins 25, 26 and 28, 29 to ground in turn. For each of these pins, thin white vertical stripes should appear on the screen. A fault in this last test indicates a fault in the shift register (74166). If everything is fine so far, we start on the memory cards.

The memory cards (fig. 5)

For now we build one memory card. The component layout is given in fig. 6. The photo shows that the 4 k RAMs are mounted in sockets; this is of course not necessary.

the editorial prototype was tested with the Intel 2107C-4 and the NEC D411D-1. With the TMS 4060 NL, inverting was not possible. A small modification has also been made to the memory cards. The 7407 IC (6 buffers, open collector) had 4 buffers to spare. These buffers were placed after the 4 data outputs. This is not really necessary; the advantage is that the RAMs are loaded slightly less. Moreover, in the event of a chip-select fault this prevents the RAMs from being damaged, since we now have open-collector outputs instead of the RAMs' three-state outputs. When the card is finished, it can be plugged (with the power switched off!) into the connector intended for it. After switching on, a random dot pattern will appear on the screen. To clear the display, briefly connect pin 13 of the interface connector to ground. If everything works so far, we can start on the interface board.



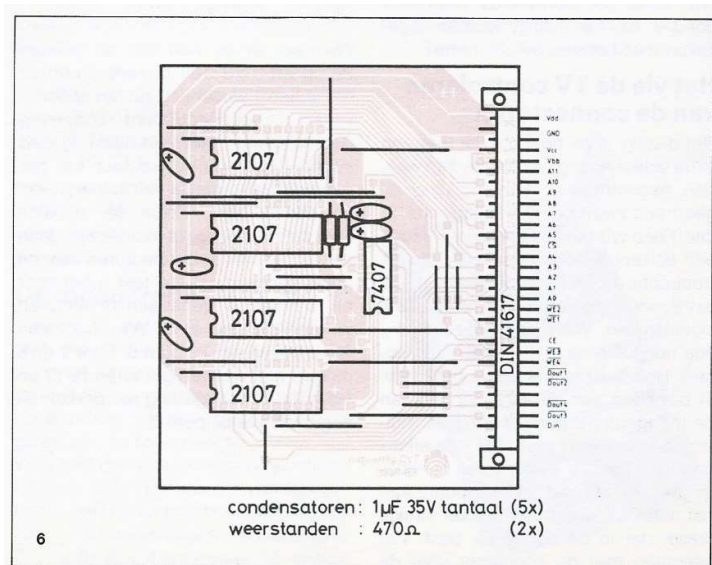
The interface board (fig. 7)

The last board is the interface board (see fig. 8), the link between computer and display. Once you have assembled this board, we are going to enter an address by hand. The memory card must be plugged into the leftmost socket. To test the card we solder wires to hv7, A1, B1, C1 and A2. Connect $\Phi 2$ to ground via a small switch. After switching on, a random pattern will appear on the TV. Lines A1, B1 and C1

determine which action is requested; lines A2 and B2 determine whether to read, invert, write white or write black. First we must clear the display. We apply code 100 to C1, B1 and A1 by grounding A1 and B1. The moment we briefly close the switch ($\Phi 2$ to ground), the display will turn black. Before removing the connections, the switch must be opened again. Now we put address \$7F into register X: code 001 loads register X. We make hv7, C1 and B1 a logic '0' and flip the switch once on and off. The same address must go into register Y. To fill this register we must apply code 010; so C1, A1 and hv7 to ground again, and flip the switch on and off. Address \$7F7F is now in the buffer. To write a white dot we must apply code 10 to B2, A2; so A2 to ground. Then the code for writing (011) must be applied to C1, B1, A1; so C1 to ground. When the switch is flipped on and off, a white dot should appear in the middle of the TV screen.

Software

In practice the above operations are done much faster by the microcomputer. In the next part we will deal with the software needed to put letters and lines on the screen. If you want to try something yourself already: the line blanking of the display is fed to the computer via the flat cable. When the computer wants to write a dot, it must wait until this sync pulse has occurred. We will also return to this in the next part.



4	hv3	hor. or vert. address
5	hv4	hor. or vert. address
6	hv5	hor. or vert. address
7	hv6	hor. or vert. address
8	hv7	hor. or vert. address
9	/ø2	clock pulse from computer.
10	A1	\ Control lines for the type of action.
11	B1	} (Clear, write, etc.)
12	C1	/
13	A2	\ Control lines for the colour to be written
14	B2	/
15	Dout	On a read action, gives the colour of the dot.
16	LB	Line sync pulse for synchronising the computer.
24	GND	Ground.

The hardware for this graphic TV display was completed in the previous 3 parts. In this part we deal in detail with the two most important routines, namely writing a dot and clearing the picture. With these routines it is possible to write the first software for the display. You can expect 2 more important programs from us: first, writing letters using a software character generator; second, drawing a straight line between two points. With these programs construction phase 1 will be complete.

The connections between computer and display

The display must be connected to the computer's I/O. Table 1 gives the display connections. We will briefly explain the purpose of these connections.

THE SIGNALS hv0...hv7

These signals (outputs, as seen from the computer) pass an 8-bit address. This address defines the position on the TV screen. Depending on the state of control lines A1...C1, the address can be a horizontal or a vertical address. The horizontal address runs from 00 (left) to FF (right). The vertical address runs from 00 (top) to FF (bottom).

THE SIGNALS A1...C1

With these three control lines (outputs, as seen from the computer) we tell the display what we want to do. We can choose between clocking in a horizontal or vertical address, clearing the whole screen, and giving a read/write command. The remaining bit combinations are not used (for the time being?). Table 2 shows which action is taken for which bit pattern.

THE SIGNALS A2, B2

These control lines (outputs, as seen from the computer) indicate which 'colour' the dot must get. By colour we mean black, white, invert or read. With invert, the dot becomes

white if it was black and black if it was white. With read, a '1' or a '0' comes back over the data line, depending on whether the addressed dot was black or white. Table 3 shows which colour belongs to which bit combination.

THE SIGNAL /Φ2

This is the only signal that does not come from the computer's I/O. It must indicate when the data on the control lines is valid. It sometimes happens that during the change from, say, 001 to 010, state 011 occurs for a very short time, wrongly generating a 'write' command. /Φ2 must be connected to a point in the system that is high during this transition. In 6502 and 6800 systems this will be /Φ2. Only when /Φ2 goes low will the data presented on the control lines be decoded.

Dout

This signal (an input, as seen from the computer) is used only for reading. Dout becomes 1 for a black dot and '0' for a white dot.

THE SIGNAL LB

By means of LB (line blanking) the computer can synchronise with the display. The computer must give its write (or read) command during the visible part of a line on the screen (LB = low). During retrace the display puts the presented data on the screen.

TABLE 1. Connections between computer and display

WIRE SIGNAL

1	hv0	hor. or vert. address
2	hv1	hor. or vert. address
3	hv2	hor. or vert. address
4	hv3	hor. or vert. address
5	hv4	hor. or vert. address
6	hv5	hor. or vert. address
7	hv6	hor. or vert. address
9	hv7	hor. or vert. address
8	/Φ2	clock pulse from computer.
10	A1	\ Control lines for the type of action.
11	B1	} (Clear, write, etc.)
12	C1	/
13	A2	\ Control lines for the colour to be written
14	B2	/
15	Dout	On a read action, gives the colour of the dot.
16	LB	Line sync pulse for synchronising the computer.
24	GND	Ground.

[Note: in this table wires 8 and 9 are printed as hv7 = 9 and /Φ2 = 8, the reverse of Table 1 in part 3 (hv7 = 8, /ø2 = 9). One of the two is probably a misprint.]

TABLE 2. Function of control lines A1...C1

C1 B1 A1 function

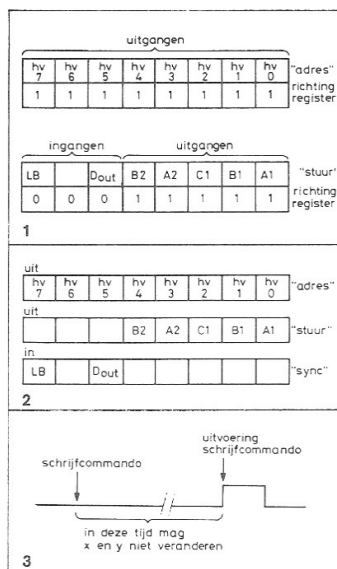
0	0	0	idle
0	0	1	clock in horizontal address
0	1	0	clock in vertical address
0	1	1	read/write addressed dot
1	0	0	clear entire picture

TABLE 3. Function of control lines A2...B2

B2	A2	colour
0	0	read
0	1	invert
1	0	write white
1	1	write black

The computer's I/O

In the rest of this article we assume that the display address is on one output port, and that the control lines plus LB and Dout are on a second port. Bits 5, 6 and 7 of this second port must be programmed as inputs. Fig. 1 shows the connections for 6800 and 65XX systems. Port A ('ADRES') is used for the addresses and Port B ('STUUR' = control) is used for the control signals and the 2 incoming lines. The settings of the data direction registers are shown below the registers. When a program is started, we must always begin by setting the data direction registers (i.e. \$FF in PADD and \$1F in PBDD). For 8080-oriented systems we follow fig. 2; here the incoming signals get a separate input port.



Writing a dot on the screen

Putting a dot on the screen involves a little more. First the horizontal and vertical addresses must be clocked in, the desired colour must be set, and the write command must be given during the visible part of a line. The command, however, is only executed during the next line blanking pulse. Fig. 3 illustrates this. New addresses may therefore only be clocked in 1.6 μ s after the positive edge of line blanking. Fig. 4 gives the flowchart of the write routine. We assume that the colour has been set once beforehand. This is easily done by putting the correct bit combination of A2 and B2 on 'STUUR'. To prevent premature actions we must set (or leave) control lines A1...C1 in the idle state ('000'). In hex we therefore get:

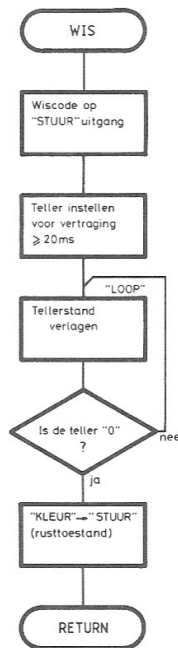
\$00 for read
\$08 for invert
\$10 for write white
\$18 for write black

This corresponds to the codes in table 3, assuming that A2 is connected to bit 3 and B2 to bit 4. The first step in the write routine is now to clock in both addresses. From now on we will call the horizontal address the X address and the vertical address the Y address. If we want to write a series of dots (which will usually be the case), we must make sure before clocking in the addresses that any previously given command has been executed. Therefore the routine begins by waiting until line blanking is high. At that moment we know for certain that any previous write command has been executed, so we can safely pass new addresses. The codes needed for clocking in the X and Y addresses and the code for the write command were deliberately chosen in numerical order, so we can always step to the next code with an 'INC' instruction (for the 8080: 'INR'). The flowchart (fig. 4) therefore begins with a wait loop. As soon as 'line blanking' is high, the X address is put on the outputs ('ADRES'). An 'INC' (or 'INR') instruction puts code 001 on control lines C1...A1, so that the address on the computer's outputs is clocked in as the X address. As long as this code is on the control lines, the display's X address buffer follows the computer's outputs exactly. Before the Y address is put on these outputs, we must increment the control lines once more with 'INC'. Now the display's Y address buffer becomes active and will initially clock in the still unchanged X address.

But, as said before, the Y buffer keeps following the computer's outputs. So we can now put the Y address on the 'ADRES' outputs (fig. 1 or 2). Applying 'INC' (INR) once more also closes the Y address buffer and automatically generates a write command. As the last action, control lines C1...A1 must be returned to the idle state ('000'). Preferably the colour should be retained, so that it is ready for writing the next dot. The fastest way to achieve this is to fetch a copy of the write colour from elsewhere in memory and put it on the 'STUUR' outputs. So when we want to change the write colour, the corresponding code must be written both to the 'STUUR' outputs and to the memory location holding the copy. The codes on the 'STUUR' outputs now successively take the following values:

	B2	A2	C1	B1	A1	
1 idle state	X	X	0	0	0	idle
2 after 'INC'	X	X	0	0	1	clock X
3 after 'INC'	X	X	0	1	0	clock Y
4 after 'INC'	X	X	0	1	1	action
5 copy of write colour to the outputs	X	X	0	0	0	idle

Lines B2 and A2 hold the bit combination for the programmed write colour throughout the whole process.



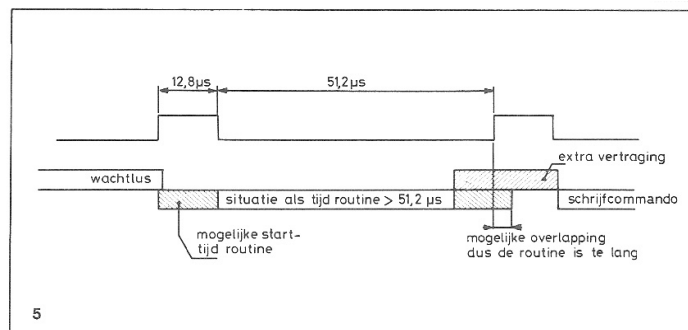
Synchronisation

Because the write command may only be given during the visible part of a line (line blanking is low), the whole process is subject to a time limit. The routine starts when the LB signal is high. Normally we start in the wait loop and then detect the positive edge of the LB signal.

It is possible, however, that we jump into the wait loop at a moment when LB is already high. In that case we 'roll' straight through the wait loop. So the actual routine can begin anywhere from the positive edge until just after the negative edge of the LB signal. In fig. 5 this region is shown hatched.

As said, the write command must be given while LB is low. This means the whole process must not take too long. The LB signal stays low for $51.2 \mu\text{s}$, so the whole routine (including one pass through the wait loop) must not take longer than $51.2 \mu\text{s}$. If the computer system used is too slow, we must make sure that the write command does not come earlier than after $76.8 \mu\text{s}$ (fig. 5).

After this time the LB signal is certain to be low for the second time. Clocking in the X and Y addresses may, incidentally, take place while LB is high.



The program

In this part we give the routine needed to put a dot on the screen. When the routine is executed as a subroutine, any figure can easily be put on the screen. In a following part we will deal with drawing letters and drawing lines between two points. The routines below have not yet been assembled, because the various memory addresses must be determined by the user. [Label/operand names are kept as in the original: SCHRIJF = write, WACHT = wait, STUUR = control port, ADRES = address port, XADR/YADR = X/Y address, KLEUR = colour, SYNC = sync input port. The last column is the execution time in μs .]

6502 (1 MHz)

SCHRIJF	LDA STUUR	4
	BPL SCHRIJF	2
	LDA XADR	3
	STA ADRES	4
	INC STUUR	6
	INC STUUR	6
	LDA YADR	3
	STA ADRES	4
	LDA KLEUR	3
	INC STUUR	6
	STA STUUR	4
	(RTS)	
total required:		45 μ s

In calculating the instruction times we assumed that the PIAs are not in page 0. The locations XADR, YADR and KLEUR were of course chosen in page 0. The total time required stays within the 51.2 μ s limit. The last instruction (RTS) is of course only needed if this routine is executed as a subroutine. The whole thing would be another 6 μ s faster if the X address were in index register X and the Y address in index register Y. In practice, however, these registers often already have another job.

8080 (2 MHz)

SCHRIJF	IN SYNC	5
	ANA A	2
	JP SCHRIJF	5
	MOV A,E	2.5
	OUT ADRES	5
	MOV A,L	2.5
	INR A	2.5
	OUT STUUR	5
	INR A	2.5
	OUT STUUR	5
	MOV A,D	2.5
	OUT ADRES	5

44.5

At this point the X and Y addresses have been clocked in. The subtotal shows that there is no longer enough time to give the write command. The LB signal can go high again after as little as 51.2 μ s, and will certainly have done so after 64 μ s. At least 2 'OUT'

instructions are needed for the 'write pulse', so the total time comes to at least 54.4 μ s. There are 2 ways to make the 'write pulse' fall in the next low period of the LB signal. The first method is to build in an extra delay: we know that LB goes high within 64 μ s and stays high for 12.8 μ s, so with an extra delay we can ensure that the write command follows only after 76.8 μ s. Method 2 is simply to watch LB and wait until it goes high; the routine then continues as follows:

```

WACHT      IN  SYNC      5
           ANA  A         2
           JP   WACHT     5
           MOV  A,L       2.5
           ORI  03H       3.5
           OUT  STUUR     5
           MOV  A,L       2.5
           OUT  STUUR     5
           (RET)

```

Again we wait for the positive edge of LB. The time that then elapses (before the OUT instruction starts the write sequence) is more than 12.8 μ s, so we know for certain that LB is low. In this 8080 program we assumed that the X address is in register E and the Y address in register D. The write colour must be in register L.

6800 (1 MHz)

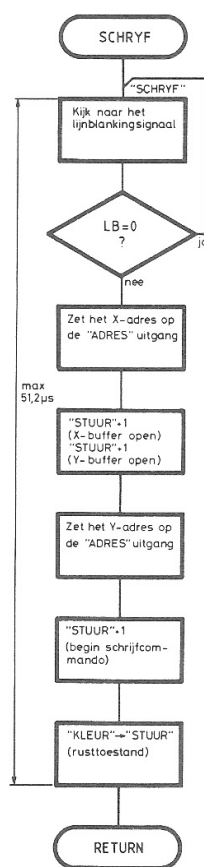
```

SCHRIJF    LDA  A  STUUR      4
           BPL  SCHRIJF      4
           LDA  A  XADR       3
           STA  A  ADRES      5
           INC  STUUR         6
           INC  STUUR         6
           LDA  A  YADR       3
           STA  A  ADRES      5
           LDA  A  KLEUR      3
           INC  STUUR         6
           STA  A  STUUR      5
           (RTS)

```

total time required: 50 μ s

Here too the PIAs are not in page 0, while the locations 'XADR', 'YADR' and 'KLEUR' are. Of course accumulator B can also be used instead of accumulator A.



Clearing the screen

The clear code (100) does not automatically produce a clean screen.

In previous parts we already wrote that we sought a balance between software and hardware. Well, in this case the hardware only makes the /WE line of all RAMs go low. When we want to clear the screen, all memory locations must become '0' or '1', depending on the desired colour. To achieve this, the clear code must be applied for about 20 ms. In this time all addresses have been run through once, and the picture is white or black. Clearing therefore does not take place specifically during retrace, and so there is no need to synchronise with the LB signal. The colour is normally set with control lines A2, B2. Because the clear code makes all 8 /WE lines low simultaneously, it is not possible to use invert or read as the 'colour', since we cannot select one of the 8 bits. We can of course make the picture white or black. The hex code to put on 'STUUR' for the various colours is:

\$14 black screen

\$1C white screen

\$04 \ undefined

\$0C / ('speckled' screen)

The flowchart of the clear routine is shown in fig. 6. After the clear code has been on the 'STUUR' output for 20 ms, the original colour must be restored and control lines A1...C1 must return to 000 (idle). As noted earlier, a copy of the colour code must always be kept in RAM. By putting this copy, which we call 'KLEUR' here, on the 'STUUR' outputs at the end of the clear routine, all control lines return to the correct position. Writing the clear routine from the flowchart (fig. 5 [sic; fig. 6]) should present no difficulties.