(Preliminary Version, 1st Revision - Sept. 1985)

Upgrading Dragon 32s to 64k of RAM
by R.W.Hall, National Dragon Users Group.

Disclaimer. Although every care has been taken in the preparation of these notes, and the technical data have been checked, neither the author nor the User Group accept any liability for inaccuracies or consequential loss or damage, however caused.

1.Introduction.

These notes are provided to help the adventurous upgrade their Dragon 32s from 32k of random access memory (RAM) to 64k, and to give them a feel for possible uses for the additional memory. The technical detail given is adequate rather than exhaustive, and assumes that you have some competence at electronics, and in the programming, an acquaintance with the hardware of the machine, such as may be found by reading say 'Inside the Dragon' (Smead & Somerville, published by Addison-Wesley, 1983). The notes are based on articles I wrote in Dragon Update, and the feedback that I received from other members of the User Group.

2.Is upgrading worthwhile?.

To answer this question let's compare the spec of the Dragon 64 with that of a 32:-

The Dragon 64 currently retails for £195; it has twice as much RAM as the 32, an RS232 interface, and two BASIC ROMs - the additional ROM is used when the 64 is in '48k' mode. A Dragon 32 by contrast retails for £50 (if you can find one), and secondhand fetches about £25-35. Changing the memory chips will cost you well under £15 if you do it yourself (£40-60 if someone else does!), (and adding an RS232 interface - following the circuit in July 85 'Electronics & Computing' - would cost probably another £15). Without the additional ROM however, the additional RAM cannot easily be used by BASIC. So, if you don't have a disk system, and always write in BASIC, upgrading to 64k is probably **not** going to be useful to you - if however you know at least a smattering of machine-code or use a language like FORTH, then even if you haven't got a disk yet, you'll probably find the extra memory useful. If of course you have got a disk drive, you'll be itching to make the mod.(?!).

3.First Steps.

The circuit-boards in D32s come in half-a-dozen different varieties, but there are two basic types - those having 8 32k (by 1bit) RAM chips, and those having 16 16k RAM chips. The former are rather easier to modify than the latter (more common) types. To find out which sort you've got, type the following, immediately after switching on:-

```
?PEEK(&HFF22) AND 4
```

If the answer is 0 then you've got an 8 by 32k system, if its 4 then your memory is 16 times 16k chips - if you get any other answer, steady your trembling fingers and try again!.

(The significance of this test bears explaining:-
when the Dragon is switched on, or when reset is pressed, a routine in ROM at (hex) B3B4 is entered, which sets up the default settings of the two Peripheral Interface Adapters (PIAs), the Video Display Generator (VDG) and the Synchronous Address Multiplexor (SAM); until the SAM has been set up correctly, the RAM cannot be used. To set up the SAM however, the ROM routine on a D32 must know the memory type, and the 2nd bit at FF22 is programmed as input and used for this purpose. The D64 reset routine does not need to know the memory type, and the bit is instead used as an output signal to switch between the two BASIC ROMs).

If you find that you have (8 times 32k) memory it is very likely that the chips are in fact 'half-good' 64k units (they are often marked as OKI 3732s).

It's fairly straightforward to check if you do have these sorts of chips, by trying to write information to the **uppermost 16k of the 64k** (ie C000-FEFF), **when there is nothing plugged in the cartridge port,** (and then trying to recover it), using the routine given in Appendix B. However, you won't be able to test ALL of the upper 32k (if it exists) until you've made the mod. to the decoder circuitry. This is, I suggest, the next step to make, in ALL cases.

4.Modifying the Decoder Circuitry.

The modification to the decoder circuit is straightforward, and can be added either externally or internally - however, it is only worth making the mod externally (via a board in the expansion port) if you find you have 'half-good' 64k RAMs aboard, you have no diskdrive or intention to get one, and are happy to discover and work with the imperfections in the memory. The internal mod uses a spare gate on one of the 74LS02s - this is the same gate as some people have used to add a 'reverse-video' (green-on-black instead of black-on-green) capability (see Update No. 5) - if you've added this you'll have to first remove it and move it to another spare gate, wired as an inverter.

The details of both the internal & external mods are given in Figs 1 & 2. Both perform a similar function - they stop the 74LS138 (a '3-to-8 decoder') from being activated when Write-to-Memory is true, and thus prevent the BASIC & cartridge ROMs being spuriously activated when writing to RAM in the memory map above 32k. This circuit is functionally equivalent to that used by the D64 and has the same side-effect, namely:-

If you have any device in the cartridge port between 8000 and FEFF which is activated by the 'CART SEL'' signal (pin 32), it will not be possible to WRITE to this device: this is because 'CART SEL' is derived from the LS138 decoder. If the device is an I/O device, then the correct solution is to move its address to FF00 or above using P2 (pin 36) and address lines as the select-device signals. (If it's static RAM, you could separately decode the upper address lines, but do you need it now anyway?; if so, see Appendix D for a possible fix).

The external mod. could be built up on any of the prototyping boards which are available from Tandy, Compusense or Windrush. However, these are fairly expensive, certainly compared with the two logic gates which make up the circuit; a cheaper option may be to use a 44 way plug and extender board for the Spectrum expansion port, and cut it down to 40 ways to fit the Dragon port.

It is possible to make the internal mod. without removing the SAM and CPU, provided you take the usual precautions against high voltages & static (earth the iron, the board and you, and wear no nylon clothes). In detail the mod is as follows:-

i) Check that IC26 is a 6821 (PIA), IC33 is a 74LS138, and IC31 a 74LS02 with the gate on pins 1,2 & 3 unused.

ii) Now join pin 12 of IC31 to pin 3 of IC31, and pin 2 of IC31 to pin 21 of IC26 (this is just a convenient R/W line); (IC35, also a 6821 PIA, may be more convenient than IC26 on some boards). Reassemble, and check all still works.

iii) Then join pin 1 of IC31 to pin 5 of IC33; and finally, cut the connection between pin 5 of IC33 and pin 8 of IC33 (this is easier than it sounds - however the hardened steel tips of a good pair of miniature side cutters will do the trick).

iv) Test out the circuitry -the effect on normal operation should be nil.

If you have 'half-good' 64k RAMs aboard, you can now begin exploring their potential.

5.Changing the RAM chips.

a) **Preliminaries.**

The first problem is to get hold of 8 64k RAM chips at a reasonable price. Chips with an access time faster than 300ns will probably be good enough, and the only real subtlety is to get hold of 128-cycle refresh chips, not 256-cycle refresh chips - this **excludes** in particular the Texas 4164 (but not the Toshiba or NEC or Motorola 4164s!). (The 6883 cannot properly refresh 256-cycle chips - although these **may** be adequately refreshed, miles out of spec., by the action of the VDG taking information from them). I give a possible list of suitable chips

and their suppliers in Appendix C - at the time of writing it should be possible to purchase suitable chips at well below £1-50 each. Even at their present low prices however, I suggest you socket the chips, since this makes fault tracing much easier - suitable low-profile 16-pin DIL sockets cost about 12p each.

Before beginning the removal and replacement of the RAM chips I would suggest you remove the SAM and CPU (which are socketed) - removal should be effected gently, using eg a thin-bladed knife to lever up the chip with as well distributed a force as possible (again, with everything earthed!). Wrap these safely away (in tinfoil or on a conducting mat for maximum safety), before turning attention to the main circuit board (- make the decoder mod. now, if you have not already done so).

There are two common designs of mainboard in D32s - these are:-

i) the 'Issue 2' board, which comes in both (8 x 32k) and (16 x 16k) versions, but with only 8 chips directly on the mainboard,

ii) the 'Mark II' board which has room for all 16 (16k) RAM chips on the mainboard.

You may well come across others, but the notes which follow should help you through these too.

b)(8 x 32k) Machines.

Machines with (8 x 32k) memory chips are usually built on 'Issue 2' mainboards, and the procedure for modifying them is straightforward - you must remove the 8 old memory chips (usually numbered IC1-8), and replace them by sockets, check out all lines and voltages, and install the new RAM chips. The way you do the removing is largely dependent on personal skill and tools available. I have used a chip extractor together with an IC desoldering attachment on a moderate-wattage iron - others swear by desoldering braid, or a solder extractor, whilst still others cut their losses and snip out the chips. In all cases take care not to use excessive force, because the lines are plated-through, and breaking the plating is a sure cause of later malfunction. Also be warned that it is very easy to bridge lines with solder splashes, especially just under the ICs, where the lines are not lacquered over and are very close together. It is for this reason that I would embark on a systematic check that all 8 address lines (Z0-7) and control lines CAS, RAS and WE between SAM and RAM are independent, and that the 8 data lines in and out on each chip are also OK (see Appendix A, & Figs). I think this test is most safely made with volts on the board, using a low-voltage high-impedance ohms range on a multimeter (NB - on some meters the highest impedance ranges use dangerously large battery voltages). (It is wise to preceed this resistance check with a voltage check, but note that lines with logic connected to them (Din, Dout, RAS and usually Z7) give non-zero voltages). If bridges are found, it is often possible to clear them, if you've bought the right design of socket(!), by prising the plastic surround away from the soldered-in pins, to inspect the state of the lines beneath.

c) (16 x 16k) machines.

On these machines the 16 RAM chips are arranged electrically in two banks of 8, which are activated by different 'Row Address Strobe' signals, RAS0 & RAS1; the chips are usually MK4116s, (or equivalents, eg MM5290s), and instead of having a single 5V supply (on pin 8), have -5v on pin 1, 5v on pin 9 and 12v on pin 8 (-check these). These voltages (especially the 12v!) now need to be rearranged (see the pinouts in Appendix C) for the 64k chips, which go in bank 0 (IC1-8) only. In addition the RAS1 signal needs bringing over from bank 1 (pins 4) to become the 8th address line (A7) on bank 0 (pins 9), and the SAM initialisation routine needs informing of this change of duty by a change of the signal on pin 12 of the PIA IC35 from high to low.

Owners of 'Issue 2' boards will find they can effect most of this by just changing over jumpers - for owners of 'Mark II' machines the path is slightly more thorny, but not as difficult as at first appears.

i) 'Issue 2' boards.

A lot of D32s of 82-83 vintage were built on 'Issue 2' mainboards. These had room for only 8 chips on the mainboard, and often 8 OKI 3732s were fitted (see above). However, they are even more common in their (16 x 16k) versions, when the 8 extra memory chips were added in sockets above the others; or else a small subsidiary 'patch' board was added, containing 10 RAM chips and the 74LS244 memory buffer - 6 RAM chips remained on the mainboard, with connections to the patchboard taking up the space normally occupied by IC1,IC8 (RAMs) and IC16 (74LS244).

To facilitate their use as either (8 x 32k) or (16 x 16k) boards, 5 jumpers were fitted, just in front and to the right of the power supply connector (looking from keyboard!). The positions of these jumpers are numbered on the board from 1 to 10 (odd positions are at back), and they perform the following functions (which, if you are in any doubt, you should verify!):-

a) The first (leftmost) jumper controls the inclusion, or not, of an inverter in the RAS1/Z7 signal line. In position 1 there is no inverter, in position 2 there is.

b) The second jumper controls the voltage on pins 1 of the RAMs. In position 3 it is -5v, in position 4 it is +5v.

c) The third controls the voltage on RAM pins 8. In position 5 it is 12v, in position 6 it is 5v.

d) The fourth controls the use of RAM pins 9. In position 7 there is +5v on the pins, in position 8 it is RAS1/Z7.

e) The fifth (rightmost) jumper controls the voltage on pin 12 of IC35. In position 9 this is 5v, and in position 10 0v.

In (16 x 16k) mode the jumpers will be set to positions 1,3,5,7,9, and there will be a flying lead from position 8 taking the RAS1 signal to the patch-board.

The straightforward way to modify these boards is as follows:-

a) Remove the patch board and its connecting lines, saving the 74LS244;

b) Remove IC2-7, C12, C58 and any other capacitors on the line connecting pins 9 of IC1-8; remove also C59 and R66 on the RAS0 line.

c) Clean up the board, and fit 16 pin sockets in positions IC1-8, and a 20-pin socket in IC16.

d) Set the jumpers to 1,4,6,8,10

e) After checking all voltages and lines, install the 8 new memory chips in IC1-8 and the 74LS244 in IC16, ... and step back.

This method of modification produces a nice clean job, with everything easily accessible in case of trouble. However the patchboard may well have its chips installed in sockets, and it is then tempting to try installing the new memory chips on this patch board. I suggest the following route, which is for an issue 1 patchboard, with the proviso that I gave up on it!:-

a) Remove IC2-7 etc from mainboard as described in b) above.

b) Remove IC1-10, R1 & C23 from patchboard.

c) Take the flying lead, labelled 'RAS1', from jumper position 8 and connect it to the wire going to pin 4 of IC1 on the patchboard (ie to pick up RAS0) - it's probably also worth linking the junction of R1 & C23 on the patchboard to the junction of C59 & R66 on the mainboard, to produce the same effect.

d) Now for the catch - unfortunately, on the issue 1 patchboard, the 5v supply to pin 20 of IC11 (the 74LS244) is chained in with the lines connecting pins 9 on IC1-10. This chain must be broken, and IC9 & 10 relinked to IC1-8. I suggest breaking the line, which runs underneath the board, under IC5 (at back edge of board, near centre); and at the front righthand edge by IC10. After reinstating a connection between pin 9 of IC10 (by deft unseeable soldering!) and the wire to pin 9 of IC1, the difficulties are (I think) over.

e) Set jumpers to 1,4,6,8,10; check out

f) Put new memory chips in patchboard sockets IC2-7, IC9 & 10, and return 74LS244 to IC11 if you removed it....now turn on, and the best of luck.

ii) 'Mark II' boards.

The sixteen memory chips are all installed on the mainboard and the two banks are usually numbered IC1-8 (bank 0) and IC36-43 (bank 1). Both sets of chips are to be removed, but only IC1-8 are replaced with sockets. (NB physically, the chips are arranged as a group of 9 and a group of 7 - IC43 is next to IC8). Because there are no jumpers to help you on these boards all the changes must be made by cutting/rerouting lines. These wiring changes are as follows:-

i) the -5v line on pins 1 should be chopped (if necessary at both ends of the bank) and left to float; I did this modification on my board, a Mark II, Issue 6, by cutting through the line on the underside of the board, near the PSU socket, but just after the connection to TR4 and R69. On most boards the -5v line is not a through-line, but in cases where it is the connection between the two ends of the chain must be reinstated with a patch wire. (Note. The pins 1 are not connected internally on most 64k RAMs - it is not nice to leave the -5v line connected however, since on some chips (eg Motorola 6665) this pin is pulled up internally to +5v!; moreover if -5v is accidently cross-connected to any of the RAM, SAM or CPU chips it will blow them!).

ii) the +12v line (on pins 8) must be similarly chopped - it in general is a through-line (en route to the cartridge port) and so the through-circuit must be completed by a tidy patch-wire.

iii) the +5v line (pins 9) is then chopped at both ends of the bank, and the voltage diverted via the old 12v line connecting pins 8. I suggest you sever the line between C12 & IC1, and again between ICs 8 & 43/C50.

iv) the old 5v line connecting pins 9 now becomes the eighth address line. Remove any bypass capacitors on the line, then take a patch-wire from the line to that connecting pins 4 of bank 1. Using the old 5V line in this way as a signal line probably loads up the signal with rather more capacitance (80pf?) than is present on the other address lines (because the land is quite wide) but this does not seem to have any adverse affect on the circuitry - note that connecting the signal across in the way suggested ensures the inclusion of the 22 ohm damping resistor (R12) between SAM and RAM.

v)when all this is done, it remains to change the signal on bit 2 of FF22. This is done by disconnecting pin 12 of IC35 (6821, PIA) from R73 (a 'pull-up' resistor connected to 5v), and wiring it to ground.

This ends the mods - a subsequent careful test of all voltages on the RAM, SAM & CPU sockets, followed by a systematic test for bridges or other errors, prior to replacing the chips, is particularly recommended (-see remarks in 5b). Testing out by software in the event of malfunction is largely fruitless, because most faults lead to totally dumb systems - however, provided you have checked that the voltage rails are correct (and in particular that the -5v and 12v lines have been routed away from the RAM), all changes should be reversible. (Note that failing to perform step v) (or equivalently on an Issue 2 board, failing to change jumper 5 from posn 9 to posn 10) **will** produce a running system but with only 16k of memory visible to BASIC.

d)Jokers in the Pack!.

In the cut-throat world of micro-manufacturing, Dragon Data seemed to have shown commendable ingenuity to use what components they could get hold of, especially RAM chips. The result is that, although the three boards described above are the most common, there exist a number of less common varieties, of which I don't have 'hands-on' experience. One of these is the so-called 'Siemens' board, built using 8 Siemens HYB-4232s. Though numbered as if 32k chips, these were in fact double 16k units. They were 18-pin chips but were installed on Issue 2 boards, and the instructions given above **should** suffice to see you through.

Secondly, there are later versions of the 'patchboard', I believe, and on some of these DD actually installed 32k chips! A careful inspection of your board (if it's not an Issue 1, dealt with above, and if it doesn't already contain **only** 8 x 32k chips, when the answer is clear!), should tell you if its

worth keeping in place or not.

In all cases of doubt, make sure you arrive at the connections between the principle components outlined in Appendix A.

6.What can be done with it all.

Let's now turn to ways of using the new memory which do not require a disk-drive.

When the Dragon is first switched on, it is in Memory-Map Mode 0, and the new 32k of RAM is tucked away out of sight. To access the new memory we must either switch to Map Mode 1, or switch from Page 0 to Page 1 in Map Mode 0. In Map Mode 1 the new memory occupies 8000 to FEFF and we have no access to the BASIC & cartridge ROMs; neither can we access the last 256 bytes of RAM (FF00-FFFF) which are reserved for I/O. In Map Mode 0, Page 1, the SAM automatically adds 32k to the cpu (but not VDG) addresses before sending them to RAM so that the new memory appears to lie at 0-7FFF, (in place of the true lower memory, which is now inaccessible). In all three modes however (Map Mode 0, Page 0; Map Mode 0, Page 1; Map Mode 1), the VDG has access to all 64k of RAM (and never to the ROM), which it sees in the 'correct' order.

It's fairly straightforward to get from Map Mode 0 to Map Mode 1, and vice versa, in machine code or FORTH (in fact any language which doesn't need the ROM routines for a bit), by toggling the addresses FFDF & FFDE (see Appendix A) - but you must turn the interrupts off first. An example of this is given in Appendix B, program 1, which is a routine to copy blocks of data between low & high RAM, or between ROM and high RAM. It may be used to copy the BASIC & Cartridge ROMs into high RAM, and then run them - it is then possible to customise the BASIC/DOS and also have easy access to the memory from E000 to FEFF.

Program 2 uses a high-res text generator (NOT provided!) resident in high-memory to write text from BASIC on to a high-res screen, also in high memory. (This routine thus effectively gives BASIC routines use of 8k of graphics RAM for nothing - note that it makes use of the fact that the VDG will continue to display from high memory, whilst the processor is in Map Mode 0.)

Access to and from Map Mode 0, Page 1, requires care, because the system stack has to be reset each time; page-switching is achieved by toggling addresses FFD4 or FFD5, again with the interrupts disabled. In program 3 the two pages are used to give two wholly independent areas for BASIC, so that two independent 32k programs can be run. There is no reason why the two programs need be wholly independent however, once the essential features of the changeover are understood.

In all these investigations of the Dragon's Memory Modes, the Motorola data sheet for the MC6883 (reproduced in 'Inside the Dragon') makes an invaluable, if concise, reference.

7.Flex versus OS9.

Many people contemplating upgrading to 64k have an eye to running either OS9 or Flex; (NB OS9 is not available for Premier 'Delta' disk systems).

Both Compusense Flex and Dragon Data OS9 operating systems will work with an upgraded machine (provided you've done the decoder mod.!) - however, whilst Flex will boot up straightaway, a slightly more roundabout method is needed for OS9, because the absence of an RS232 port at FF04-8 causes the bootstrap routine to hang up. To get round this, proceed as follows:-

i)type 'BOOT' - the words 'OS9 BOOT' should appear on the screen, and a few seconds later all disc activity ceases, but with the disk still selected and turning.

ii)press Reset.

iii)type 'POKE &HFF03,&H34: EXEC9736'.

Success should then be yours.

A brief comparison of the advantages of the two systems may be useful.:-

Flex is structurally simpler than OS9, being essentially a single-user, single task operating system of mid 70's vintage; it has been described by Mike James (in 'The 6809 Companion') as 'almost without argument the standard operating

system for the 6800 and certainly the best single-user operating system for the 6809'. By contrast, OS9 is rather complex, being multi-user and multi-tasking; the multi-user facility is largely academic for Dragon users, and the main use of multi-tasking is to run a 'background job' to the printer whilst proceeding with something else on the screen - there just isn't enough memory to do much else anyway. ('Printer-spooling' is a feature of some implementations of Flex, but Compusense decided not to implement it on the Dragon for rather purist reasons). The Flex system comes with a 51 by 24 character set, with a facility for 'windows'; the 51 by 24 character-set is optional in OS9 (type 'GO51' to get it) but when it is installed, there is only about 40k of userspace free, as against 48k for Flex. Where OS9 scores is on the availability of cheap(ish) software - even before the recent price cuts by Touchmaster and Cotswold Computers the OS9 packages were more attractively priced that their Flex counterparts, and with the recent dramatic price reductions this is now even more true. (This is rather paradoxical, since one of the claimed advantages for Flex is the wealth of software available for it - there is, from specialist firms like Windrush, if you'll pay £300 a time!). The only readily available software for Flex is the editor/assembler, DBASIC, RMS (also available under OS9), and a wordprocessor. The assembler is quite nice (it allows 'macro's'), but not perfect (eg. it doesn't optimise the code for PC-relative forward references); the editor is an oldfashioned line-editor, as is the the OS9 one; DBASIC is essentially a copying-over of the Microsoft ROM into RAM - it has its uses but has none of the speed or aids to structured-programming of OS9's BASIC09. (And I haven't tried the Flex word-processor, but this is written using OS9 Stylograph). Overall, I feel that the Flex Editor/Assembler is a very powerful package for developing machine-code programs, but OS9 has more to offer in other lines (eg C & Pascal), if you can live with its memory limitations. At the end of the day it's the old tale of horses for courses - why not play safe and buy both??!!!

8.The Future - 256k?

256k RAM chips are now the same price 64k chips were 18 months ago (£3-50 or so). These chips have the same pinout as the 64k ones, except that pin 1 is now used for a 9th multiplexed address line. A recent article in BYTE (September 85, pages 247 to 254) showed how such chips could be incorporated on the Atari 800XL, and how the refresh problem (all 256k chips are 256-cycle refresh) could be overcome; it is likely that a similar method could be applied to the Dragon (or 'CAS before RAS' refresh could be implemented). The outstanding problem is how to manage the memory map (the Atari already has a dedicated register for this), and who will produce software to run on a 256k machine. A suitable 'Rolls-Royce' quality memory management unit for the 6809 does already exist - it's called the Motorola 6829 and costs about £25. If you're prepared to install one (NOT trivial!) you could in principle have a machine capable of running OS9 Level Two (if you can persuade someone to sell this you). Ordinary mortals however, would be advised to wait awhile, until Messrs Compusense & Eurohard sort out their own upgrade path. If you socketed your chips, you can wait safe in the knowledge of being prepared for the next leap forward.

9.And Finally...

I'd like to acknowledge helpful feedback from Paul Grade, John Bussell, Alan Butler, Arne Eriksonn, and John Payne. I hope that you find these notes useful - if you feel you've got a problem however, or if you just want to talk some aspect of this upgrading through, write to me via Paul, or phone me on ~~0272-294113~~ *(evenings). Your comments will be very welcome - in keeping the Dragon a viable machine, we're all in this together!

* 0454-416445

B S Hall.

APPENDICES.

Appendix A. Memory Control on The Dragon.

The MC6883/74LS783 Synchronous Address Multiplexor or SAM plays the central role in the architecture of the Dragon, interfacing the control of the RAM, ROMs, 6809E cpu, and VDG, and providing the E and Q clocksignals. All this is accomplished in a standard IC package of only 40 pins, and the penalty for this is that, having no connection to the cpu data bus, the SAM must receive its orders from the cpu on the address bus. 32 addresses, from FFC0 to FFDF, are used for this purpose, and control 16 bits of information; writing (anything!) to the odd addresses sets a bit, whilst writing to the even addresses clears it. Most well-known of these operations is the so-called 'speed-up poke' (POKE&HFFD7,0 and its reverse, POKE&HFFD6,0); however, of more concern to us here is the Map Type Bit (TY), set by FFDF, cleared by FFDE; the Page Bit (P1), set by FFD5, cleared by FFD4; and the Memory Size Bits (M0 & M1) set/cleared by FFDB/FFDA & FFDD/FFDC respectively. Of the remaining addresses, FFD6 to FFD9 control the 2 clockrate bits (R0 & R1); FFC6 to FFD3 control 7 bits (F0 to F6) giving the base address of the current Video-RAM (in units of 512 bytes); and FFC0 to FFC5 control 3 VDG Mode bits (V0 to V2).

The SAM interleaves requests from the cpu for data, with providing data for the VDG. Depending on the map mode and the memory address sent out by the cpu, the SAM will either pass on the address required to RAM, or will activate a ROM or output device, (via signals sent to the LS138 decoder); meanwhile, the address of the next byte in video-RAM for output to the VDG is generated in the SAM itself.

There is more in passing an address to RAM than meets the eye, because to reduce the size and complexity of the RAM chips, the address is presented in two parts, called the row address and column address; this is the 'address multiplexing' process for which the SAM gets its name. When 32k or 64k chips are in use, the SAM first presents the lowest 8 bits of the address to the RAM chips on its output lines Z0 to Z7, and asserts the Row Address Strobe (RAS); about 0.1 microseconds later (!) it presents the upper half of the address on the same lines, and asserts the Column Address Strobe (CAS) - about 0.1 to 0.2 microseconds later again, the data output from the RAM chip has stabilised, and is read on the data bus (-or, if write-enable (WE) is activated, data is read from the bus into memory). Now when 16k RAM chips are used, only 7 lines (Z0 to Z6) are needed for addressing; Z7 is now programmed (via M1) to act as a second Row Address Strobe (RAS1), thus allowing the use of two banks of 16k chips.

Finally, the SAM controls the 'refreshing' of the RAM chips. Dynamic RAM chips have a simple structure of one transistor and capacitor per bit (or cell), and the charge leaks away from the capacitor over a few tens of milliseconds. The RAMs are so constructed however that everytime a row address is presented to them, all the cells with this row address have their information read out into buffers - later in the cycle this information (or new information if there is a write-to-memory) is read back into the capacitors, so that the whole row is refreshed at once. The SAM inserts a series of these 'RAS-only refresh cycles' from time-to-time to keep all the cells updated. As a final subtlety, not all 256 RAS combinations need to be addressed on most chips, only 128. However some chips (notably the Texas 4164) do require 256 refresh addresses, and cannot be refreshed by the 6883.

Layout of the Dragon.

Both Dragon & Tandy machines closely follow the recommended circuit given by Motorola in the MC6883 data sheet. On all the boards the SAM output lines (Z0-Z6, RAS0, RAS1/Z7, CAS & WE) are linked to the corresponding rails on the RAMs (A0-A6, RAS, or A7, CAS & WE) by low-value damping resistors; on the 'Issue 2' boards however, there is optionally also an inverter between Z7 and A7, presumably to allow the good upper half of 'half-good' 64k RAMs to be used where appropriate; it can safely be left in circuit when upgrading. As for the data lines on the chips, the inputs Din are connected directly to the data bus from

the cpu, whilst the outputs Dout (marked as 'Q' on most RAM datasheets!) go via a 74LS244 buffer on to the data bus, and via a 74LS273 latch to the VDG; data is strobed into the latch under control of RAS0, whilst the 74LS244 memory buffer is activated by a signal from the 74LS138 decoder (which in turn decodes signals S0,S1,S2 from the SAM - see block diagram, Fig 4). Note that, on MkII & Issue 2 mainboards, IC1 is connected to data line D7, and so on in reverse order til IC8, which is connected to D0.

Appendix B. Some programming examples.

The three programs given in listings L1 to L3 have already been briefly introduced in the main text. At the heart of each are a series of short machine-code routines; these are coded in hexadecimal and placed in BASIC DATA statements, (one instruction per statement). The surrounding BASIC program loads the the machine code, by default into the cassette buffer at (hex) 200, and then where necessary modifies it, or installs jumps in the appropriate BASIC input/output hooks. As a rule these m/c routines perform their manipulations with the interrupts turned off - this is because in normal operation the interrupts are vectored, by jump instructions in locations (hex) 100 to 112, into routines in ROM; unless both the vectors and service-routines are copied over to RAM the system will crash when an interrupt occurs. The most frequent interrupt comes from the 50Hz clock, which can be turned off separately, in BASIC, by executing 'POKE&HFF03,&H34'.

Program 1: Here the machinecode begins by saving all registers and turning interrupts off : it then executes a simple loop in which data is picked up from an address pointed to by X, in one map mode, and deposited in the address given in the Y register, in another map mode. The map mode for picking up the data, for depositing it, and the final map mode to return to, are all set up by BASIC, as are the start & finish addresses in the first field, and the start address in the second. By copying from 8000 to E000 (if you have a disk, BFFF if not) in mapmode 0, to 8000 on in mapmode 1, and returning to mapmode 1, you will smoothly transition from running BASIC in ROM to running it in RAM, giving space for machinecode etc from E000 (C000) to FEFF.

Program 2: This uses the Compusense Flex High-Res text-generator resident in RAM above 32k to write text from BASIC to a high-res graphics screen at E600 in RAM. The text has true lower-case, and a 51 by 24 format. (A similar arrangement could probably be made with other similar text-generators.)

To use the program first boot Flex (not provided!) and turn off any options set for the terminal (eg unset 'pause at end of page' by typing 'TTYSET, DP=0'), then type 'MON' to return to normal BASIC. Finally load and run the program - the BASIC may be deleted after it has run, leaving the m/c routines.

The program works by intercepting the BASIC 'output a character' hook; if the output is heading for the disk, cassette or printer (9D nonzero) then it is sent on its way - (the version shown assumes you have DragonDos, so amend line 40 to 39,12,12 if not). If the output is heading for the screen, then the map mode is changed to 1 and the Flex 'output a character' routine or the 'output a crlf' routine entered; (the addresses of these and other useful Flex routines are to be found in the Flex 'Advanced Programmer's Guide'). Finally, it is necessary to keep the VDG looking at the right graphics screen, since there is a BASIC routine called everytime a character is input which resets it to look at the normal text screen at (hex) 400. A simple modification is hooked into the input ram hook to achieve this.

Program 3: This program uses page-switching, to run two independent BASIC or machinecode programs in the two 32k pages of memory. The program is configured to run with DragonDos, and to remove this feature change line 140 (JMP $D917) to '140 DATA 39,12,12'.

The code comprises 3 sections: SWOPIT, which dumps the stack pointer in "HOLE", switches pages, and reloads the SP from the corresponding location in the other page; VDG, which keeps the Video Display looking at the right text-screen (400 in page 0, 8400 in page 1); and finally MOVIT, (a derivative of

program 1), which sets up Page 1 initially, by copying page 0 into it and then modifying some locations.

After running the BASIC, type NEW to remove the redundant code, but leave the machinecode. Typing or executing 'EXEC' will enter SWOPIT, and send you to the other page. SWOPIT could also be linked into the 'End-of-BASIC-line' hook at 19A-19C, or at a pinch into the 50Hz interrupt routine. The routine will work correctly (?!) for all BASIC programs which use text only, and m/c routines which use the BASIC I/O. It will also generate graphics correctly, but in Page 1 will not display them unless the BASIC statement 'SCREEN a,b' is expanded to 'SCREEN a,b: POKE &HFFD3,0' (and equivalently for m/c programs).

Use of FORTH. Those wishing to use FORTH will find it easy to change map/page modes by defining words such as C64, thus:-

```
HEX : C64 0 FFDF C! ;
```

However, you will need to turn interrupts on and off - the following FORTH word, CLOCKOFF, would usually be sufficient,

```
HEX : CLOCKOFF 34 FF03 C! ;
```

but a more satisfactory solution is to define e words IOF and ION in machinecode as follows:-

```
HEX CREATE IOF
1A50 , AEA1 , 6E94 , SMUDGE
CREATE ION
1CAF , AEA1 , 6E94 , SMUDGE
DECIMAL ;S
```

(This screen should work with Oasis/Dragon Data & Tele- Forths. The m/c AEA1, 6E94 translates as LDX ,Y++ :JMP (,X) and is the 'in-line ' code for the FORTH terminating-word 'NEXT').

Appendix C. Some suitable 64k RAM chips and their suppliers.

Manufacturer	Number	Possible Stockist**	Telephone
Oki	M3764	Manhattan Skyline	0628-75851
Hitachi	HM4864	Farnell	0532-636311
Motorola	MCM6665	STC	0279-26777
Mostek	MK4564	Celdis	0734-586191
NEC	PD4164	Farnell	0532-636311
Toshiba	TMM4164	Verospeed	0703-644555

All of these have the same pinout (see fig C2), with pin 1 being marked as N/C, except for the MCM6665, for which the pin is connected internally to 5V; also given in the figure (C1) is the pinout of a typical 16k memory chip (MK4116), usually fitted to the Dragon 32.

All the 64k chips above are 128-cycle refresh types (in contrast to the Texas 4164 which is a 256-cycle chip and is **not** suitable).

**Finally, the stockists listed above are (apart from Manhattan Skyline) often not the best sources for small quantities . The following four firms,
Technomatic (01-208-1177)
Midwich Computer Co. (0379-898751)
Watford Electronics (0923-37774)
Happy Memories (054-422-618)
have recently been offering suitable chips at very competitive prices (eg less than £1 a chip for HM4864s at Happy Memories). Check the latest position in 'Wireless World' & 'Electronics & Computing'.

Appendix D. An alternative decoder mod., allowing use of static RAM in the cartridge port.

The decoder modification suggested in the main text is the logical one to use following the Motorola data sheets, and is essentially the same as that employed on the Dragon 64. However, it does have the disadvantage of not easily

allowing static RAM in the cartridge port, because the cartridge-select line is disabled during all write operations. It appears that the following alteration to the internal mod. would overcome this:- instead of gating the decoder logic (IC33,74LS138 & IC31,74LS02), with R/W (the read/write line from the processor), the signal WE (write-enable) is used. To do this, connect pin2 of IC31 to the land joining pins 3 of the RAM chips (rather than to pin 21 of IC 26); the other connections, between LS02 and LS138, remain the same.

It must be emphasised that this is a rather speculative mod., using an undocumented property of the WE signal - namley, that WE does not go low when memory above 7FFF is addressed in Map Mode 0 - this has been determined by experiment only! For this reason, I do not recommend this mod. Having said this, it would be a great convenience to have static RAM available in this way when using the page-switch facility, since the system satck and a data interchange area could be maintained in it (contrast Program 3 of AppendixB).

BASIC

```
10   'MOVIT 23-JAN-84
20   DATA 34,7F
30   DATA 1A,50
40   DATA 8E,00,00
50   DATA 10,8E,00,00
60   DATA B7,FF,00
70   DATA A6,80
80   DATA B7,FF,00
90   DATA A7,A0
100  DATA 8C,00,00
110  DATA 2F,F1
120  DATA B7,FF,00
130  DATA 35,FF
140  '
150  PRINT"movit here":PRINT"GIVE ME A HOLE TO LIVE IN";: INPUT BA
160  IF BA)300 GOTO 170 ELSE BA=512
170  FOR I=BA TO BA+30
180  READ V$:V=VAL("&H"+V$)
190  POKE I,V
200  NEXT I
210  PRINT"TAKE DATA FROM";:INPUT ST,FI
220  PRINT"IN MAP MODE 0(32K) OR 1(64K)";: INPUT MI
230  PRINT"AND PUT IT AT";: INPUT NA
240  PRINT"IN MAP MODE";: INPUT MO
250  PRINT"RETURN TO 32K(0) OR 64K(1) MODE";: INPUT MR
260  IF MR=0 THEN POKE BA+28,&HDE ELSE POKE BA+28,&HDF
270  IF MI=0 THEN POKE BA+13,&HDE ELSE POKE BA+13,&HDF
280  IF MO=0 THEN POKE BA+18,&HDE ELSE POKE BA+18,&HDF
290  X=INT(ST/256): Y=ST-256*X
300  POKE BA+5,X: POKE BA+6,Y
310  X=INT(FI/256): Y=FI-256*X
320  POKE BA+22,X: POKE BA+23,Y
330  X=INT(NA/256): Y=NA-256*X
340  POKE BA+9,X: POKE BA+10,Y
350  EXEC BA
```

ASSEMBLER

```
PSHS   X,Y,A,B,DP,CC,U   Dump the lot
ORCC   £$50              Interrupts off
LDX    £0000             Nasty impute instruction
LDY    £0000             And this
STA    $FF00             Will be FFDF or FFDE
LDA    ,X+               Pick up Data
STA    $FF00             Will be FFDE or FFDF
STA    ,Y+               Dump Data
CMPX £$0000              Another modified address
BLE    L                 If less go round loop
STA    $FF00             Final map mode here
PULS   X,Y,A,B,DP,CC,U,PC   All done.
```

PROGRAM 1.

BASIC

```
10  'FLEXIT 8-MAR-85
20  DATA 0D,6F
30  DATA 27,03
40  DATA 7E,D9,17
50  DATA 32,62
60  DATA 34,7F
70  DATA 1A,50
80  DATA B7,FF,DF
90  DATA 81,0D
100  DATA 27,0A
110  DATA BD,CD,12
120  DATA B7,FF,DE
130  DATA 8D,07
140  DATA 35,FF
150  DATA BD,CD,24
160  DATA 20,F4
170  'CHANGE VDG PAGE
180  DATA 34,02
190  DATA 86,F0
200  DATA B7,FF,22
210  DATA B7,FF,C5
220  DATA B7,FF,C3
230  DATA B7,FF,C0
240  DATA B7,FF,D3
250  DATA B7,FF,D1
260  DATA B7,FF,CF
270  DATA B7,FF,CC
280  DATA B7,FF,CA
290  DATA B7,FF,C9
300  DATA B7,FF,C7
310  DATA 35,82
320   '
330  CLEAR 500
340  PRINT"flexit here":PRINT"GIVE ME A HOLE TO LIVE IN";: INPUT BA
350  IF BA<300 GOTO 360 ELSE BA=512
360  FOR I=BA TO BA+73
370  READ V$:V=VAL("&H"+V$)
380  POKE I,V
390  NEXT I
400  X=BA:Y=INT(X/256):Z=X-256*Y
410  POKE &H169,Z
420  POKE &H168,Y
430  POKE &H167,&H7E
440  X=BA+35:Y=INT(X/256):Z=X-256*Y
450  POKE &H16C,Z
460  POKE &H16B,Y
470  POKE &H16A,&H7E
```

ASSEMBLER

```
Start   TST    $6F        Check ouput device
        BEQ    SCRN       If 0 it's O/P to screen
        JMP    $D917      If anything else send to DOS hook
SCRN    LEAS   2,S        Drop return address off stack
        PSHS   X,Y,A,B,U,DP,CC
        ORCC   £$50       IOF
        STA    $FFDF      Map mode 1
        CMPA   £$0D       CRLF?
        BEQ    CRLF
        JSR    $CD12      Char O/P
FINISH  STA    $FFDE      Map mode 0
        BSR    RESET
        PULS   X,Y,A,B,U,DP,CC,PC
CRLF    JSR    $CD24
        BRA    FINISH

RESET   PSHS   A
        LDA    £$F0
        STA    $FF22      Set VDG mode
        STA    $FFC5      Now set
        STA    $FFC3      VDG
        STA    $FFC0      Base page
        STA    $FFD3      to
        STA    $FFD1      $E600
        STA    $FFCF      which is where
        STA    $FFCC      the FLEX
        STA    $FFCA      HIRES
        STA    $FFC9      textscreen
        STA    $FFC7      lives
        PULS   A,PC       All done
```

PROGRAM 2

BASIC

```
10  'HOLE-STACK STORED HERE
20  DATA 0,0
30  'SWOPIT - SWITCHES PAGES
40  DATA 34,7F
50  DATA 1A,50
60  DATA10,EF,8C,F6
70  DATA 32, 8C,75
80  DATA B7,FF,D4
90  DATA 10,EE,8C,EC
100 DATA 35,FF
110 'VDG - CHANGE VDG PAGE
120 DATA 7D,00,6F
130 DATA 27,03
140 DATA 7E,D9,17
150 DATA BD,80,0C
160 DATA 32,62
170 DATA 34,16
180 DATA 8E,FF,C8
190 DATA A7,0A
200 DATA 7E,A9,41

210 'MOVIT - SETUP INITIAL CONFIG
220 DATA 34,7F
230 DATA 1A,50
240 DATA 8E,00,00
250 DATA 10,8E,80,00
260 DATA B7, FF,DF
270 DATA A6,80
280 DATA A7,A0
290 DATA 8C,7E,FF
300 DATA 2F,F7
310 DATA 86,D5
320 DATA A7,8C,C6
330 DATA 4A
340 DATA A7,8D,7F,C1
350 DATA 86,0A
360 DATA A7,8C,D6
370 DATA 4C
380 DATA A7,8D,7F,D1
390 DATA B7,FF,DE
400 DATA 35,FF
410 '
420 CLEAR 500
430 PRINT"swopit here":PRINT"GIVE ME A HOLE TO LIVE IN";:INPUT BA
440 IF BA)300 THEN BA=512
450 SWOPIT=BA+2:VDG=BA+22:MOVIT=BA+45
460 FOR I=BA TO BA+92
470 READ V$:V=VAL("&H"+V$)
480 POKE I,V
490 NEXT I
```

ASSEMBLER

```
HOLE    FDB   0            Stack stored here
SWOPIT  PSHS  A,B,X,Y,U,DP,CC
        ORCC  £$50         IOF
        STS   HOLE,PCR
        LEAS  SWOPIT+$80,PCR   In case of NMI?
LS1     STA   $FFD4            Flip page
        LDS   HOLE,PCR
        PULS  PC,A,B,X,Y,U,DP,CC
VDG     TST   $6F              Find if O/P to
        BEQ   LV2                screen
        JMP   $D917            DragonDos hook
LV2     JSR   $800C            Put char on scree
        LEAS  2,S          Drop return address off
        PSHS  X,B,A          stack
        LDX   £$FFC8       SAM VDG bits start
LV1     STA   $A,X
        JMP   $A941        Now rejoin ROM Reset
                             routine

MOVIT   PSHS  A,B,X,Y,U,DP,CC
        ORCC  £$50
        LDX   £$0000
        LDY   £$8000
        STA   $FFDF            Mapmode 1
LM1     LDA   ,X+
        STA   ,Y+
        CMPX  £$7EFF
        BLE   LM1
        LDA   £$D5
        STA   LS1+2,PCR
        DECA
        STA   LS1+$8002,PCR
        LDA   £$0A
        STA   LV1+1,PCR
        INCA
        STA   LV1+$8001,PCR
        STA   $FFDE            Mapmode 0
        PULS  PC,X,Y,A,B,U,DP,CC
```

PROGRAM 3

```
500 EXEC SWOPIT
510 X=VDG:Y=INT(X/256):Z=X-256*Y
520 POKE &H169,Z
530 POKE &H168,Y
540 POKE &H167,&H7E
550 EXEC MOVIT
560 EXEC SWOPIT
570 EXEC SWOPIT
```

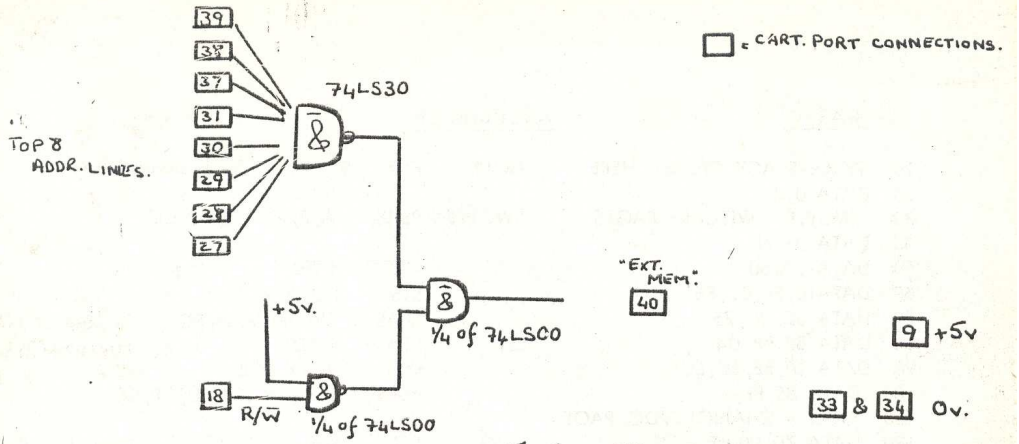


FIG1. EXTERNAL MOD.

DECODER MODS.

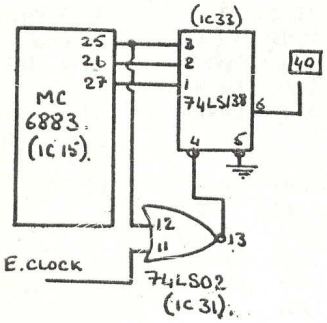


FIG2. (BEFORE).

ALTERNATIVE INTERNAL MODS.

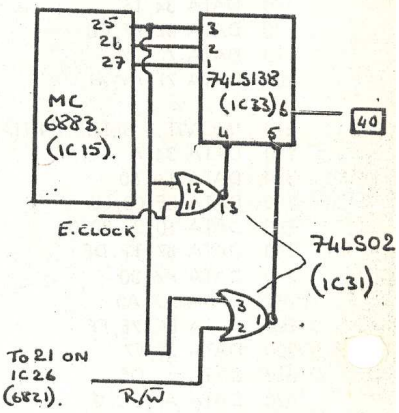


FIG3. (AFTER).

PINOUTS of 16K & 64K DRAMS VIEWED FROM ABOVE.

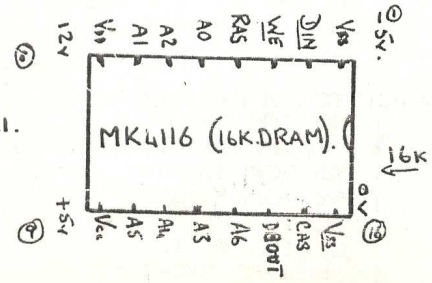


FIG.C1.

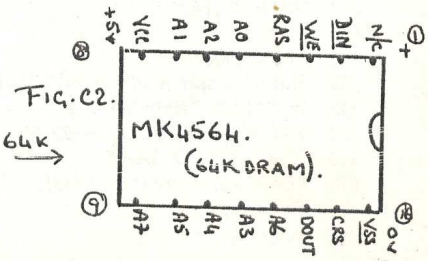
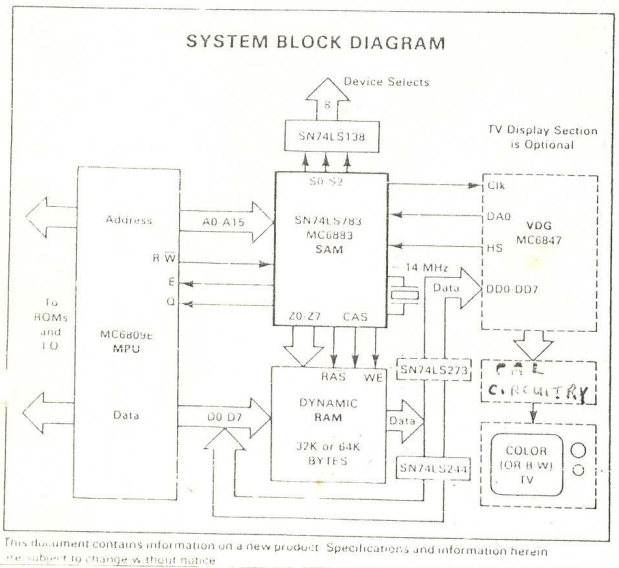


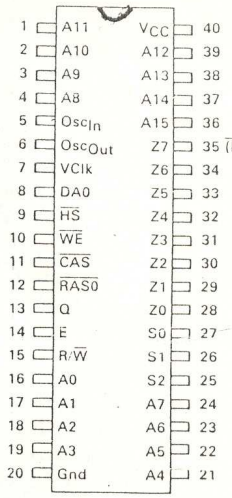
FIG.C2.

Fig 4. System Layout, & Pinouts of Principal Components



This document contains information on a new product. Specifications and information herein are subject to change without notice.

MC6883/SN74LS783 PIN ASSIGNMENT

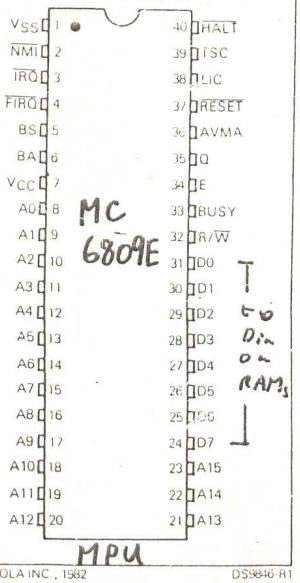


To address lines on RAMs

SAM

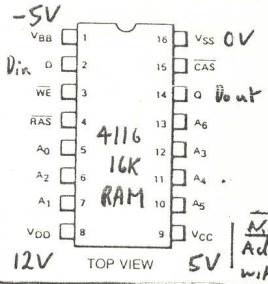
ADI-595
(Replacing NP-118)

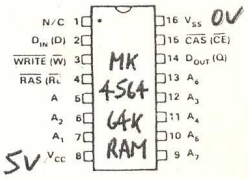
PIN ASSIGNMENTS



To Din on RAMs

MOTOROLA INC., 1982 DS9846-R1



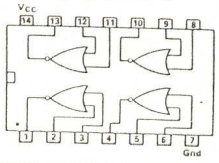


NO Don't confuse the multiplexed Address Lines A0-A6, A7 on the RAMs with the main (non-multiplexed) lines

A0-A15 between MPU & SAM

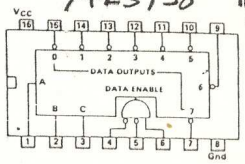
02 Quadruple 2-input NOR gate

74LS02



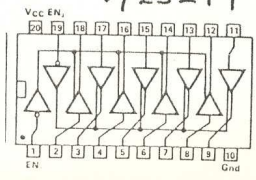
138 3 to 8 line Decoder/Multiplexer

74LS138



244 Octal buffer – three state non-inverting

74LS244



273 8-bit register with clear

74LS273