

128k RAM on the 16-bit BUS

This document is based on the writeup of Mike Ballmann's 32K on the 16-bit bus Project, as written up by John Clulow. There is an excellent graphical depiction of the original modification at Mainbyte.com.

Here, my goal is to add some additional explanation about how the system works, and optionally install 128k instead of 32k.

Now, that may seem exciting up front. My personal opinion is that THIS approach of installing 32k is more of a gimmick, honestly. There are better approaches that we can and should use. However, I did it, it works, and some people are curious about how! :)

The original document was published in the Lima Newsletter in February 1988. This document is published at HarmlessLion.com in November 2009 by Mike Brent, aka Tursi.

The original project installed two 32k RAM chips into the system in order to get the full 16-bit bus. However, due to the memory layout, the two chips overlapped the standard TI memory space, and the non-RAM area was not usable. Fairly simple mods could enable "supercart" RAM at >6000 and even non-switched "DSR" RAM at >4000, but these were never developed (and would interfere with cartridges and the PEB/Sidecars unless they could be turned off.)

You may wonder how a 32k RAM chip sees the full 64k address space. The reason is that each chip sees only half of the space – either the even addresses, or the odd addresses, thus 32k is enough.

Due to the way the TI memory map is laid out, though, half of this space is unused for RAM, being reserved for DSR, cartridge, devices, ROM, etc. This is a complicated issue to work around which is probably why the rumored CRU interface to the rest of the memory didn't materialize. The best solution is a memory mapping circuit – this solution trades off simplicity for memory space and is a good trade.

The modification offers a performance increase by disabling the wait state generator when accessing the new RAM. Later modifications (available at Mainbyte.com) show how to add a switch which can turn the wait states back on, allowing the system to run at a normal speed as well.

My 128k enhancement works by attaching the spare address lines to the 9901 using the two spare CRU bits. In this way, the entire memory space is switched with a single CRU command, allowing 4 banks of 32k, or a total of 128k, to be accessed. As I have noted, this is not ideal. Because the

entire memory space pages, you will need to use ROM or scratchpad to manage your program during the page swap, and it's uncertain whether any software may misbehave and tweak these 'unused' CRU bits, which would cause a crash. A more standard system, like AMS, would be better to investigate going forward. This modification is presented for curiosity and educational purposes.

You will need two Hitachi HM62256LP-12 RAMs, or equivalent 32k x 8 static RAM (MainByte suggests 43256LP-12). The TI memory cycle time is on the order of 600nS, so pretty much any RAM will do. To do my 128k mod, use 128k x 8 RAM chips, I used the M5M51008BP, but this is probably a more difficult to find part these days. Note that RAM chips are sold in two designations – bits, and words. The 32k chip will either be “32k x 8” or “256kbit”, and the 128k chip is either “128k x 8” or “1megabit / 1024kbit”. Data sheets are available with a little effort searching Google – make sure whatever you choose matches the pinout of the 62256 (at least for the bottom pins, in the 128k version).

You'll also need a 74LS21 (Dual 4-input AND gate) and a 74LS153 (Dual 4-input multiplexer). These can be obtained from various electronics supply houses – Digikey is a good one. All wiring should be done with wire-wrap wire. You should use a low wattage soldering pencil with a fine pencil type tip.

The modification is done on the main board of the Black & Silver console. Although probably compatible with the QI console, nobody has laid it out for you. Visit Mainbyte.com for the best pictures and diagrams, if you need help.

1. Remove the board from the console and identify the two ROMs. They are located between the GROM connector and the 9900 IC. One is parallel to the 9900 and the other is perpendicular to it. They are U610 and U611 on the Component Location Diagram. They should be labelled 830x or 831x, and are wide 24-pin ICs.
2. Bend the pins on the RAM IC's closer so they will firmly contact the ROM pins when piggy-backed. One way of doing this is to place the RAM on its side on a table and then move the body of the IC toward the table using a gentle rocking motion to bend the pins uniformly. The idea is to make for a snug fit.
3. Bend out the following pins on both 32k RAMs: 1 2 20 22 23 26 27 28. If using Mainbyte's diagrams – pay attention that he has drawn pin 1 at the BOTTOM. These pins will NOT be soldered to anything on the ROMs. Holding the IC with the notch up and looking at the top, pin numbers start

with pin 1 on the upper left, go down the left side, then across the and up the right side. Pin 28 is opposite pin 1 on the end with the notch.

On the 128k RAM, the physical pins that you need to bend are the same, but because the IC is 2 pins longer, all the numbering changes. You also bend out all four extra pins, so the new list is: 1, 2, 3, 4, 22, 24, 25, 28, 29, 30, 31, 32. For the 128k RAM only, you may clip off pin 1, it is not connected.

All the other pins, that we have not bent, nicely match the pinout of the original TI ROMs, so we will be “piggybacking” on them to get most of the signals without having to run wires. This is faster to do and more reliable as well. Since the ROMs are read-only devices, and they respond to a different address than we want the RAM to, it's not perfect, so the exceptions have to be done by wire.

4. Place one RAM over the ROM that is parallel to the 9900. Make sure the notch points toward the 9900 and that the writing on the 9900 and the RAM can be read from the same direction. Place the RAM such that pins 1 2 27 and 28 (or for the 128k RAM, 1, 2, 3, 4, 29, 30, 31, 32) extend beyond the end of the ROM. (Do note here – the 128k RAM extends quite far and rests on the 9900 – it gets a little snug in here. You may want to test fit step 6, and decide which order you prefer to do them in.) The un-notched end of the RAM should line up with the un-notched end of the ROM. There should be a sort of “spring tension” that clamps the RAM pins onto corresponding ROM pins below it. This will help to insure good solder joints. If the RAM doesn't fit tightly, remove it and bend the pins closer.

5. Solder all RAM pins not bent out to the ROM pins below. Use a low wattage pencil with a fine pencil type tip. Inspect each solder joint carefully in good light under magnification.

6. Place the second 62256 on the ROM that is perpendicular to the 9900. The notch on the RAM points away from the 9900 and toward the edge of the board. As above, solder and inspect all pins that were not bent out. (The same pins will stick over the edge – for the 128k RAM it may be necessary to GENTLY bend down the white/grey metallic strip that runs along there – only do so as much as is necessary to not press against the RAM chip, and try not to distort its shape).

7. Bend out the 74LS21 pins 1 2 4 5 6 8 10 12 14. Note that pins 1 and 14 are across from each other on this 14 pin IC.

This is going to sit on a 74LS138 already on the motherboard – that chip is a 1 of 8 decoder. It takes in the three highest address lines (A0-A2), and from that outputs a low signal on a different pin for each 8k range of memory – that is, there is a pin each for bases >0000 (ROM), >2000

(RAM), >4000 (DSR), >6000 (Cart), >8000 (Devices), >A000 (RAM), >C000 (RAM) and >E000 (RAM). We are interested in the bases >2000, >A000, >C000, and >E000, as these are the locations where the 32k RAM expansion exists. The console itself does not hook these signals up, rather the memory expansion in the PEB has another 74LS138 that re-decodes the address bus.

The 74LS21 is a dual 4-input AND gate. We use it to combine the four RAM signals into one for the RAM chip enable. Since the enable is active low, and these lines are also active low, an AND gate gives us the behavior we need, that is, high if all 4 lines are high, and low if any of the four lines are low.

(Note a potential hack here... if the AND gate is extended to include the lines for >4000 and >6000, then the RAM will *also* answer at the DSR and cartridge ROM space. In this implementation, one of the AND gates has only one of its four inputs in use, the rest are tied to VCC so that they are always high. The DSR usage is only useful if you do not have the PEB or any sidecar peripherals attached, since it's always mapped, but may be useful for standalone use or DSR development. The ROM usage would give you the equivalent of SuperCart memory built into the system. You would want a switch to disable both of these since otherwise you could never use real peripherals or cartridges with ROM, like XB or Parsec! Note that when the switch is off, the lines must be pulled high again, a double-throw switch would do this most easily.)

Because the pins of the 74LS138 don't line up nicely with the 74LS21, we need a couple of wires to get the signals we want. This particular implementation uses both sides of the 74LS21, but that is not necessary (it just worked out handy, since unused inputs need to be tied to something for reliable operation anyway. It may also have been for future use, see the hack idea above. I have also successfully substituted a quad-AND gate which is cascaded appropriately.)

8. The 74LS21 will be piggy-backed on the 74LS138 U504. this IC is located adjacent to the end of the board where the edge connector is. There are two 138's next to each other. U504 is the one nearest the end of the board. You will place the 74LS21 so that the UN-NOTCHED end lines up with the un-notched end of the 138 (pointing toward the cassette connector). Pins 1 and 16 of the 138 will extend beyond the notched end of the 74LS21.

9. Before positioning the 74LS21, solder $\frac{1}{2}$ " lengths of wire-wrap wire to the 138 pins 7 and 9. Then position the 74LS21 on top of the 138 and solder all pins not bent out to the 138 pins below and inspect the connections.

10. Bend out all of the 74LS153 pins EXCEPT 8 and 16. For the location chosen for this chip, only power and ground are available, the rest will need to be wired.

11. Place the 153 over U613, a 74LS194. This IC is below roughly the middle of the 9900, and should be immediately to the left of the 9904 (assuming the 9900 is above it). The notch will line up with the 194 notch and point toward the edge of the board away from the 9900. Solder pins 8 and 16 of the 153 to pins 8 and 16 of the 194 below.

12. At the end of the 9900 opposite to where the RAM's have been piggy-backed, you will see a line of three ICs. They are 74LS00, 74LS32, and 74LS04. The 74LS00 is U606 and the 74LS32 is U605. Turn the board upside down so you can see the traces. Find the trace that runs from pin 11 of the 74LS00 (U606) to pin 13 of the 74LS32 (U605). Double check to make sure you're doing the pin numbering correctly. When you've found the trace, cut it with a knife so there is no continuity between the LS00 pin 11 and the LS32 pin 13. This is a short little trace that doesn't go anywhere else!

The 74LS153 is a dual 4-bit multiplexer. It takes in two 'address' bits and four 'input' bits. The address selected controls which input bit is mapped to the output. This circuit uses just one side of it, so is treated as a single multiplexer and the second side is ignored.

The cut trace took me a while to figure out, as far as I can determine this line controls the system wait state generator, when low, the multiplexer will respond to this memory request (halting the CPU and collecting both bytes), otherwise it ignores the cycle.

The 74LS153 is wired rather simply. Inputs 0, 2, and 3 are tied high (signal '1'), and input 1 is tied low ('0'). The enable input is permanently grounded, so the gate is always enabled. Input select 0 comes from the 74LS00 on the board, and represents the original signal to control the multiplexer. Input select 1 comes from the RAM chip select signal that we generated on the 74LS21. This arrangement means that when the RAM select is high (ie: NOT our new RAM chips), the original signal from the 74LS00 passes through. When the RAM select is low (ie: the RAM is active), the output is always high and the wait state generator is disabled.

The wait state generator is a little more complicated than that, but this explanation suffices for what we are trying to achieve here. Check out Thierry Nospikel's page <http://www.nospikel.com/ti99> for the fully nitty-gritty details.

13. Identify the piggy-backed RAM that is perpendicular to the 9900. Solder wire-wrap wires connecting every bent out pin on this RAM to the

corresponding bent out pin on the RAM that is parallel to the 9900. Pin 1 to pin 1, pin 2 to pin 2, etc. There will be eight wires in all to solder. (Eleven on the 128k RAM chip – you can ignore pin 1).

14. BE CAREFUL HERE IF YOU ARE DOING THE 128K RAM VERSION. DOUBLE-CHECK THE NUMBERING! Solder wire-wrap wires to make the following connections on the RAM that is parallel to the 9900. Pin 1 (128k – Pin 3) goes to pin 24 of the 9900 (solder the wire to the 9900 pin on top of the board). Pin 2 (128k – pin 4) goes to the 9900 pin 22. Pin 20 (128k pin 22) goes to two places. Connect pin 20 (128k – pin 22) of the RAM to pin 22 (128k – pin 24) of the RAM and also to pin 8 (bent out) of the 74LS21. There should be three wires coming off pin 20 (128k – pin 22) of the RAM. Pin 23 (128k – pin 25) of the RAM goes to pin 21 of the 9900. Pin 26 (128k – pin 28) of the RAM goes to 23 of the 9900. Pin 27 (128k – pin 29) of the RAM goes to pin 61 of the 9900 (fourth from the top on the right side). Finally, connect pin 28 (128k – pin 30) of the RAM to pin 20 of the 74LS244 (UJ10) adjacent to the piggy-backed 74LS21.

For the 128k RAM chips, you will still have 3 unconnected pins – 2, 31, and 32. Connect pin 32 to pin 30 – this is VCC. Pin 30 on the 128k RAM is an extra chip select, so we are just tying it permanently high.

Pins 2 and 31 are the extra address lines, which are used here to 'bank' or 'switch' the memory. I did a very simple solution that attached these pins to the 9901's extra CRU bits, which allows CRU control of the bank selected. These are CRU addresses >0020 and >0022, and connect to the 9901 on pins 37 and 38. These pins are not guaranteed to be low on startup – I have seen reports that one is high on startup and the other left low. I have observed them both low in MiniMemory, but they should never change dynamically once up. So far they appear safe. But there are few issues with this approach.

First, the paging swaps the entire memory expansion all at once, all 32k changes, meaning that the only safe CPU RAM in the system when paging is scratchpad, which is very small, or VDP, which you can't execute from. A system that controls the paging with more resolution, such as per 8k block, would be more useful but would take additional hardware, this was a quick solution.

The second is a smaller issue – any program that manipulates these bits will lose all of its RAM immediately, and probably crash. No program SHOULD be messing with them, but it can be hard to tell if one does. Using an inverter (as a buffer) and an LED to display the state of these two lines would make it obvious if someone switches them. (Conceptually in Mainbyte's step 3, but I have not broken it out myself.)

Third is that these are the only spare CRU bits in the console, and many other hacks use them to. If you happen to run any software that expects one of these hacks, it will swap memory pages instead and so may crash.

You could alternately use switches to enable/disable the paging – this would allow you to disable paging when running software that doesn't like it. For this you would use a double-pole/double-throw switch. Connect the centers to the two switch pins, connect one side to the CRU pins, and connect the other side direct to ground to tie the pins low. When tied to ground, the RAM will not bank, when tied to the CRU it will. There is a good chance the noise from switching would crash a running console, so switch with the power off.

15. Connect the following 74LS21 pins with a bare wire: 1 2 4 and 14. Connect the short wire from the 138 pin 7 to the LS21 pin 5 (bent out). Connect the LS21 pin 6 to LS21 pin 12. Connect LS21 pin 8 (bent out) to the piggy-backed 153 pin 2. Connect the short wire coming from the 138 pin 9 to LS21 pin 10. Finally, connect the 74LS21 pin 14 to the 74LS244 pin 20 that you connected the RAM pin 28 (128k – pin 30) to.

16. OK, we're almost done, so take a break and have a beer or Coke.

17. On the 153, connect pin 9 to pin 13 on the 74LS32 (U605). Pin 10 of the 153 goes to pin 14 of the 74LS74 next to it (U607). Also connect pin 10 of the 153 to pin 15 of the 153, and then connect pin 15 of the 153 to pin 7 of the 74LS00 U612 (next to the 74LS74). Connect pin 14 of the 153 to pin 11 of the 74LS00 U606; that's the one you cut the trace on.

That's it! Now have another beer or Coke before putting your computer back together. When you try it out, remember that this version isn't compatible with any other 32K in the system – so remove the card from your PEB. The fastest way to check is to plug in Extended BASIC, start it up, and type SIZE. If you see 24488 BYTES OF PROGRAM SPACE FREE, you win!

If you hear any beeps, smell any smoke, or it doesn't come on immediately, power off and check ALL your connections carefully.

If you did the 128k version, you can test it in MiniMemory, after the XB test is done. Load EasyBug, and you can try the following commands:

?MA000

MA000 = 00 -> 55

MA001 = 00 -> AA

MA002 = 00 -> 1

MA003 = 00 -> 2

MA004 = 00 -> 3

MA005 = 00 -> 4

```
MA006 = 00 -> 5
MA007 = 00 -> . (press period)
?MA000
MA000 = 55 -> (press enter)
MA000 = AA -> (press enter)
MA000 = 01 -> (press enter)
MA000 = 02 -> (press enter)
MA000 = 03 -> (press enter)
MA000 = 04 -> (press enter)
MA000 = 05 -> . (press period)
?
```

The above sequence does a quick test of RAM at >A000, the beginning of the high RAM bank. You can do this for both 32k and 128k if you like. You might also test >2000 (low RAM), >3FF0 (top of low RAM), or >FFF0 (top of high RAM). The idea is, we typed in some values, then went back and read the values back to make sure they were right.

To change the page, enter CRU mode. The CRU bits 20 and 22 control the pages, so:

```
?C0020
C0020 = 00 -> 1
C0021 = 01 -> (press enter)
C0022 = 00 -> 0
C0023 = 01 -> . (press period)
?
```

Note that the values you read at 0020 and 0022 might be different. 0021 and 0023 should mirror the values you enter. There are four possible combinations: 00, 01, 10, and 11. For each one, if you repeat the above memory test and use different numbers, you should find that the memory is addressable and you can save and recall different values in each bank. You should try all four combinations. Using different numbers will help when you go back to a bank after changing a different one, to see that it remembered the old numbers.

See also:

<http://www.mainbyte.com/ti99/>
<http://www.nouspikel.com/ti99/>
<http://harmlesslion.com/>