

# EEVblog Electronics Community Forum



A Free & Open Forum For Electronics Enthusiasts & Professionals

Welcome, **Guest**. Please login or register.  
Did you miss your activation email?


Forever

Login with username, password and session length

This topic

[Home](#) [Help](#) [Search](#) [About us](#) [Links](#) [Login](#) [Register](#)

EEVblog Electronics Community Forum » Electronics » Beginners » Z80 single board memory bank switching



**Used N9917A FieldFox Handheld Microwave Analyzer, 18 GHz**  
Limited offer **55% off**, options addable, calibrated

**SHOP NOW >>**

**KEYSIGHT TECHNOLOGIES**

**5 PCBs for \$2, any color**

Production of PCB, SMT, Stencil is back to normal

**J@LC JLCPCB**

« previous next »

Pages: [1] 2 3 Next All [Go Down](#)

[PRINT](#) [SEARCH](#)

**Author**

**Topic: Z80 single board memory bank switching (Read 33642 times)**

0 Members and 1 Guest are viewing this topic.

**MrAureliusR**

Supporter



Posts: 366

Country:

frozenelectronics.ca



**Z80 single board memory bank switching**

« on: July 26, 2013, 06:18:38 pm »

Hi there.

So I've been fiddling with my Z80 for a while now, and I'm ready to kick things up a notch. My goal is to get the project to the point where I can 'talk' to my SBC over a serial terminal. In order to get there, I have to load the Z80 monitor into EEPROM (done) and then of course, hook up the SRAM, EEPROM, PIO/SIO/UART (depending on which I need), memory bank selector, and MAX232 (if needed, I can't remember if the UART/SIO/PIO puts out RS-232 voltages, I don't think they do).

My understanding of the theory is pretty good, it's just translating that into practical set-up that I'm finding difficult. There's a wealth of information on z80.info and I've been reading a LOT from that site, plus all the datasheets for the chips I'm using. And I've found a few great schematics for basic Z80 systems that are helping a lot.

However, I haven't been able to find much info on actually getting everything set up. My chips, of course, are not the exact same that are in the schematics. My INMOS 4K SRAM, for example, has only 4 I/O pins, and 12 address pins. This is completely different from most of the schematics, which typically show the 62256 for RAM and the 16C550 UARTs. Also, there's not a lot of info on implementing the PIO and SIO where needed.

I'm certainly not asking anyone to design this for me -- that would defeat the whole point. What I need help with is deciding which logic chips I need, and where. I'm trying to learn and build this on my own so I'm trying to take things in small bite sizes.

Attached is an annotated photo of what I have so far. Notice that nothing is hooked up yet, as I'm still researching how to do it. Currently all the Z80's data pins are going to ground, so it's just doing NOPs over and over, incrementing the address bus LEDs. This was set up just so I could make sure the Z80 I bought was in good working order. The reset circuit works, and I have a 555 creating a 12.5Hz clock signal currently. I also have a 20MHz oscillator for when things are working properly and I'm ready to push things to their speed limit. Of course the 555 is also adjustable which is great.

So, in order to hook up both the EEPROM and either the INMOS 4k or the Toshiba 64K SRAMs, should I simply connect the address lines to both chips and then have the 5th address line (in the case of the INMOS, which only has 4) going to the chip enable on the bigger chips? So that in theory, I can separate the SRAM and EEPROM into different memory space? This is the part where I get confused. Should I have A0-A3 going to the SRAM, then A4-A15 going to the EEPROM, with A4 also going to CE on the EEPROM? Or something like that? But then, how do I connected RD and MEMRQ? Do I use a logic gate so that when it goes above 4K on the address bus it switches to the SRAM?

Oh boy, I'm feeling a bit over my head, even though I know this should be really simple!

Any and all help is much appreciated! Even just a prod in the right direction 🤔



z80\_single\_board.jpg (1407.76 kB, 1037x1555 - viewed 1514 times.)

Logged

Amateur Radio operator VA3XMR

lapm

Frequent Contributor



Posts: 557

Country:



### Re: Z80 single board memory bank switching

« Reply #1 on: July 26, 2013, 06:46:29 pm »

I think you have address line operation confused in your head. Z80 only has 64k Address space. What i would recommend, use two or three highest (A13, A14, A15) address line's to drive some sort of decoder to generate those chip select signals.

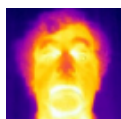
Not sure about other requirements Z80 might have since have newer used one on my own experiments.

Logged

Electronics, Linux, Programming, Science... im interested all of it...

PA0PBZ

Super Contributor



Posts: 4273




### Re: Z80 single board memory bank switching

« Reply #2 on: July 26, 2013, 06:49:41 pm »

(Trying not to give away the information, just pointing you in the right direction)  
First you have to find out where the Z80 is going to read the first instruction after a reset, because that is where you want your program to start and I'm assuming where you want the EEPROM.

Secondly, all chips need their address lines to start with A0, otherwise your program

Country: 

will not run (if it is the EEPROM), or you will going to have a hard time to store/read the SRAM.

Not sure what you mean by only 4 I/O pins for the INMOS, but you will need 8 bits SRAM, so either use 2 chips or use the Toshiba.

A PIO is a Parallel Input/Output, so that's not going to do any RS232, and yes, you will need a MAX232.

You can do simple address 'decoding' by using the highest (A15) address line so you will split the addressable memory in 2 chunks of 32KB, one for the EEPROM and one for the SRAM.

Well, enough to look at and read for now I guess.

 Logged

Keyboard error: Press F1 to continue.

 **MrAureliusR**

Supporter



Posts: 366

Country: 

frozenelectronics.ca



### **Re: Z80 single board memory bank switching**


« **Reply #3 on:** July 26, 2013, 07:06:30 pm »

Okay, you're right, I had it a bit mixed up. I forgot that it's A15 you use for simple switching, you simply connect all the others on the bus and then switch A15 on or off to indicate which chip it's going to, basically.

Yeah, the INMOS has 12 address pins and 4 I/O pins. Attached is the datasheet. I think I will probably use the one of the Toshiba's, but either way they're both SRAM so they're good for low clock speed. Plus no refresh circuitry!

I think I'm starting to get it. The Toshiba's have an active low Output Enable pin, and then two Chip Enables, CE1 which is active low, and CE2 which is active high. I'm getting things a bit tangled up in my head again, though. There's also a R/W control pin. Would MREQ go to the low CE1, and then A15 to OE, and then tie CE2 high, attach both RD and WR to the R/W pin, but put the correct one (whichever one is asserted by active low) through an inverter? Is that making any sense?

[EDIT: OE is also active low, so I guess A15 would have to be inverted as well?]

 [IMS1423P-45\\_SRAM.pdf](#) (287.71 kB - downloaded 306 times.)

 Logged

-----  
Amateur Radio operator VA3XMR

.....

 **C**

Super Contributor



Posts: 1345

Country: 

### **Re: Z80 single board memory bank switching**

« **Reply #4 on:** July 26, 2013, 08:05:01 pm »

This direct from Zilog could answer your questions

Z80 Family CPU User Manual - Zilog

the zilog manuals for the other Zilog chips are also good at explaining what is needed.

You do not have to start from scratch on software. You could copy the drivers for your SIO & PIO from a CPM-80 system that used these parts  
CP/M Plus was fancier and was able to use more memory by bank switching

and there was turbo-dos that was a network of master & slave z80's  
Using the ideas from turbo dos you could have your Z80 running CPM-80 programs while your SBC is acting like a terminal, disk drive & printer.

For some of the turbo-dos slaves they were not much more then the Z80 & ram, the master did a big assist in the boot process. Your SBC might be able to do this also

C


 Logged

 **MrAureliusR**

Supporter



Posts: 366

Country: 

frozenelectronics.ca



### **Re: Z80 single board memory bank switching**

« **Reply #5 on:** July 26, 2013, 08:40:57 pm »

---

**Quote from: C on July 26, 2013, 08:05:01 pm**

---

This direct from Zilog could answer your questions

Z80 Family CPU User Manual - Zilog

the zilog manuals for the other Zilog chips are also good at explaining what is needed.

You do not have to start from scratch on software. You could copy the drivers for your SIO & PIO from a CPM-80 system that used these parts  
CP/M Plus was fancier and was able to use more memory by bank switching

and there was turbo-dos that was a network of master & slave z80's  
Using the ideas from turbo dos you could have your Z80 running CPM-80 programs while your SBC is acting like a terminal, disk dirve & printer.

For some of the turbo-dos slaves they were not much more then the Z80 & ram, the master did a big assist in the boot process. Your SBC might be able to do this also


C

---

It's funny you should mention this -- my ultimate goal is to have CP/M 2 or 3 running. That's a bit of a ways away, I'd have to implement some sort of IDE interface. I know people have implemented IDE with a CompactFlash card slot, and that seems like a good idea. Even SD -- as I have a ton of SD cards lying around that'd be great.

I actually have a ROM of a Z80 monitor that this guy wrote for his single-board Z80 and he posted it online with a free license. So I compiled that and it compiled perfectly using the Z80 assembly compiler I have. Then I programmed that into the Atmel EEPROM I have on the board there. I've been trying to match his setup at least to some degree so I don't have to mess around with his code too much (as I don't know assembly well at all). I suppose I could if I had to... All his work can be found [here](#).

In the beginning of his monitor code, he has a memory map to show how the memory space is divided between EEPROM and RAM. Does this look like a good starting point or should I maybe just jump to CP/M? I have the sources for CP/M, I just need to compile them, which shouldn't be too hard.

I suppose I need to figure out my physical setup first, before I get into software. Or is it the other way around? 

 Logged

---

-----  
Amateur Radio operator VA3XMR

### **Re: Z80 single board memory bank switching**

« **Reply #6 on:** July 26, 2013, 09:50:38 pm »

---

Not sure where to start. There are, however lots of schematics on the net for Z80 based systems. If you'll excuse the fact that I haven't played with one for 20 years


 **grumpydoc**

Super Contributor





Posts: 2705

Country: 

(though there are increasing numbers accumulating in the parts bin so I really should put together a few simple systems sometime) here are some random thoughts.

CP/M is mildly tricky on a Z80 because it wants RAM at low addresses, unfortunately the Z80 pushes you towards ROM at low addresses because reset pushes 0000H into the program counter (actually that's pretty much all reset does) so you tend to want ROM there just after reset.

A minimal Z80 system can be built with a static RAM, a ROM and a few latches for a bit of memory mapped I/O. Depending on the RAM&ROM size you can construct a basic address decode with something like a 74LS138 (to decode 8 blocks of 8K) or half a 74LS156 (to decode 4 16K blocks). Using the former as an example you connect the top three address bits ( $A_{15}$ ,  $A_{14}$  and  $A_{13}$ ) to the  $A_2$ ,  $A_1$  and  $A_0$  inputs and MREQ to one of the active low enable inputs and that gives you 8 select signals which you can connect to the active low chip select lines on your EPROM and RAM. Obviously if you have one 8K EPROM and one 8K RAM you'll have six unused selects. These can be used for "memory mapped" I/O.

The INMOS RAM chip you mention is a 4kx4 so you need two of them one connected to  $D_0$  to  $D_3$  and one to  $D_4$  to  $D_7$  but really it's a lot simpler to find a 6264 somewhere.

True bank switching is very easy (but I'd get a system without going first) - I use two 74LS189 16x4 SRAMs to create a 16x8 RAM. The address inputs are connected to  $A_{12-15}$ , the 8 data out lines become the expanded address bus lines  $A_{12-19}$ . The 8 data in lines connect to the data bus. CS can be tied low and WE needs to be gated from IORQ, an I/O address of your choice and the CPU WR line.

The beauty is that (apart from the address decode) you basically just need the two RAM chips to expand the address bus to 20 bits - to program remember that "OUT (C), A" is really "OUT (BC), A" so you use the top 4 bits of the B register to select which RAM address to write to - no need for otherwise rather complex gating to select between address lines depending on whether you're programming or using the bank switch.

The snag is that it's static RAM so will start up with random contents - thus you need to design your hardware such that following reset your ROM takes precedence until you flip a bit to enable the bank switching (after programming the mapping RAM, obviously).

---

#### Quote

have the sources for CP/M, I just need to compile them, which shouldn't be too hard

---

You need to write a BIOS to match your hardware (BT, DT)

Or build your hardware to match an available BIOS.

You pretty well need 64K of RAM for CP/M

---

#### Quote

Should I have  $A_0-A_3$  going to the SRAM, then  $A_4-A_{15}$  going to the EEPROM, with  $A_4$  also going to CE on the EEPROM? Or something like that?

---

Hmmm, definitely a bit of confusion here about how the address bus works.

Sticking with "memory" (ROM&RAM) the low order address lines go in parallel to all the devices (typically  $A_0$  to  $A_{11}$  or  $A_{12}$ ) the high order address lines are then used to derive select lines which pick just **one** of the devices for a given address. For ROM&RAM you also need to gate in the MREQ line to make sure that the (active low, usually) select line is only active when MREQ is active.


For I/O you use the same scheme except that officially you use IORQ and A<sub>0</sub> to A<sub>8</sub> to select from 256 possible I/O devices - however there are really 64K I/O addresses because the OUT (C), reg instruction actually puts the contents of the B register on the top 8 address lines.

 Logged


Super Contributor



Posts: 1345

Country: 

### Re: Z80 single board memory bank switching

« Reply #7 on: July 26, 2013, 11:09:50 pm »

The basic hardware like grumpydoc talks about is not bad cost wise. Where you start going down hill fast with rising costs is when you try to create something like the old CPM computer. The hardware & software to talk to a hard drive has got a lot more complicated.

The way around this is to not even try. Just talk fast to a PC and let it be all IO, the hard drives, printers and display. A turbo-dos slave which can run CPM programs can be very simple, a lot of ram & high speed interface. where turbo-dos can network between Z80's nothing stops you from using the same idea with CPM. The CPM hard drive is then just a file on the PC

c

 Logged

### MrAureliusR

Supporter



Posts: 366

Country: 

frozenelectronics.ca



### Re: Z80 single board memory bank switching

« Reply #8 on: July 27, 2013, 02:23:03 am »

I actually have a few 74LS138's sitting around. I've got a ton of logic chips actually, but that seems to be a popular one for this use. I do understand how that one works quite well so that's good. As I said I have two 64K Toshiba SRAMs, and a bunch of EEP and EPROMS. The EEPROM I have is 256K. I also have two TI EPROMs that are both blank (I don't have a UV box), I can't remember but I think they're both 256K as well.

So, to start, I should use one of the Toshiba 64K RAM, and the 256K EEPROM it sounds like. Unfortunately I'm literally JUST starting to scratch the surface of Z80 assembly, but I know I have to pick it up as I go or else I'll get nowhere for the most part. I'm trying to base my work off of existing free software to make it a bit easier.

Let me see if I'm getting this right. I take the top three address bits **from the Z80**, namely A<sub>13-15</sub>, and connect those to the A<sub>0,1,2</sub> inputs on the 74LS138 (pins 1, 2 and 3). I then use MEMREQ on either \E<sub>1</sub> or \E<sub>2</sub> (pins 4 or 5) to tell it when to decode the given address (A<sub>13-15</sub>) into a single active high from the Y outputs. This output from Y<sub>0-7</sub> then goes to the CE pins on the Toshiba and Atmel EEPROM? ie, let's say Y<sub>0</sub> to the EEPROM and Y<sub>1</sub> to the SRAM. Then, to select Y<sub>0</sub> (which is active low), according to the table in the datasheet I'd pull E<sub>1</sub> and E<sub>2</sub> low (one of which is being pulled low by MEMRQ, the other is always tied low, and E<sub>3</sub> is tied high), then A<sub>0</sub>, A<sub>1</sub>, and A<sub>2</sub> would all be low, which makes Y<sub>0</sub> low and Y<sub>1-7</sub> high. To make Y<sub>1</sub> low (which would deactivate the EEPROM and enable the SRAM) I'd then have A<sub>0</sub> high, A<sub>1</sub> and A<sub>2</sub> low.

Is that all correct? Sorry for the ramble and the stupid questions, I just need to make sure I have the theory right in my head before I give it a try. I think I've got it now. ~~My other question is, do I still have A<sub>0</sub>, 1 and 2 connected to both the EEPROM and SRAM? Won't the signal I'm sending to the LS138 get picked up on the address bus to the chips as well? Or do you just assume that and address things differently? That's what I'm still confused about.~~ Wow, I just re-read that and now I feel really

stupid, lol.

Here's the [datasheet I'm looking at for the 74LS138](#). Page 4 has all the tables to select which output is high. I've also noticed that if you pull E1 or E2 high, or E3 low, all the Y0-Y7 go high as well. This makes sense as this would put ALL the chips in a standby state. So the second MEMRQ is no longer asserted, all the memory goes to 'sleep'. I now understand why it's called a 3-to-8 decoder as the A0, A1, and A2 outputs are basically the number for the Y output in binary. 000 is Y0, 001, is Y1, and so on. For some reason I didn't notice that before.

Anyway, I think I'm on the right track now. Thank you everyone for your wonderful help!! I can't wait to get this thing going and have my serial communication working. I have a whole pile of MAX232's, plus as I mentioned both a PIO and SIO. What chips do I need and in what configuration to get a serial console going with a monitor? As I also mentioned I have a basic Z80 monitor that someone wrote, just for basic I/O operations, and it can also do file open, and dumps, etc. If I remember correctly, it can also compile basic assembly programs, or something like that. I know it has an IDE interface support written in, as the guy who wrote it was using CompactFlash over an IDE interface to load everything.

« Last Edit: July 27, 2013, 02:43:52 am by MrAureliusR »

 Logged

-----  
Amateur Radio operator VA3XMR

 **westfw**

Super Contributor



Posts: 3162

Country: 

### **Re: Z80 single board memory bank switching**

« Reply #9 on: July 27, 2013, 05:29:35 am »

Grumpy's advice is spot on.

The first thing you need to do is decide what your "memory map" is going to look like, like "from address 0 to 0x7FFF I'm going to have 32k of RAM, and then from 0xE000 to 0xFFFF I'm going to have 8k of EEPROM."

CP/M will want mostly RAM, starting at 0. (Did CP/M ever support a standard bank switching scheme?) A more stand-alone "controller" like application will want more ROM and less RAM.

You definitely need to start with 64k or less of memory ("bank switching" in this area was a scheme to allow more than 64k of memory to be attached to processors that would only address 64k. You can have multiple "banks" of RAM or ROM chips within 64k, but that's just "address decoding", and not "bank switching.")

Your memory chips are PROBABLY rated in number of BITS of storage; a 27256 EPROM is 256k bits of storage, organized as 32k bytes of 8 bits each.

You may be confused by the fact that memory chips are now "large" compared to the 64k address space you're trying to fill. Most of the CP/M days had chips that were 8kbytes or less (2764 EPROM, 6116 RAM (2k\*8!)), so a lot of the designs you see will be about cascading those chips and decoding lots of upper address bits into "many" individual chip selects. You may be looking at the opposite problem - taking a chip with MORE than 64kbytes of storage, and somehow munging the chipselects (and hardwiring address lines) to only use PART of the storage on the chip. Wasting memory like that may sound depressing, but it IS the way to start out!


 Logged

 **C**

Super Contributor



Posts: 1345

Country: 

### **Re: Z80 single board memory bank switching**

« Reply #10 on: July 27, 2013, 05:38:34 am »

You have right idea with 74LS138.

You just need to put it together with the Z80. The Z80 has 16 bit address, you are dividing that chunk up in to 8 equal size smaller chunks. For easy thinking, use the



address ranges of the Z80 for the outputs of the 138.

one of the fanciest Z80 systems was a TRS model II and there are Schematics online. used a bunch of 138's

Any thing that acts like a rom will work for the Z80 to boot from. if your EEP are 8 bit data they will work.

---

#### Quote

What chips do I need and in what configuration to get a serial console going with a monitor? As I also mentioned I have a basic Z80 monitor that someone wrote, just for basic I/O operations, and it can also do file open, and dumps, etc. If I remember correctly, it can also compile basic assembly programs, or something like that. .

You would have to match the hardware to that software or change that software to match your hardware.

One SIO should let you use a terminal program on a pc to talk to the Z80 monitor.

---

#### Quote

I know it has an IDE interface support written in, as the guy who wrote it was using CompactFlash over an IDE interface to load everything.

This can get expensive very fast. Would be easier, cheaper and better to have the Z80 just talk to a PC program that acts like the disk drives & printer. Do you do any programming on a PC?

C


 Logged

 **grumpydoc**

Super Contributor



Posts: 2705

Country: 



**Simple Z80 memory interfacing.**

« **Reply #11 on:** July 27, 2013, 02:28:16

pm »

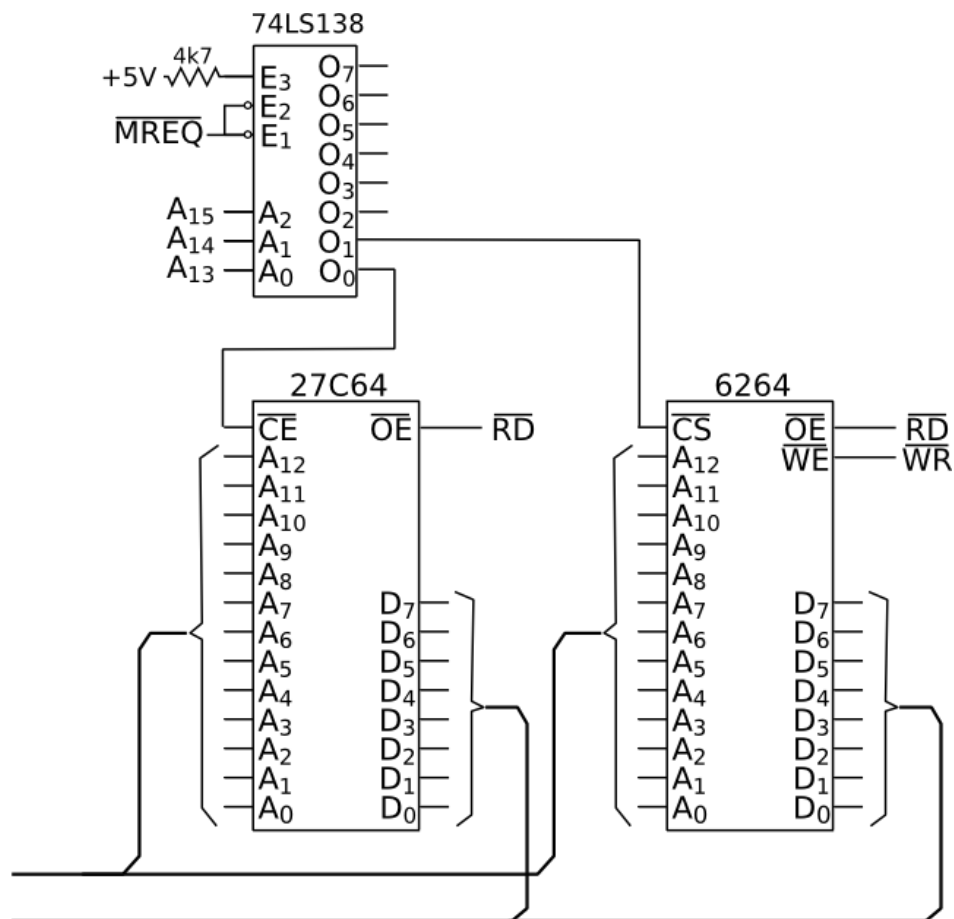
Still somewhat in the running from memory stakes and I know the OP said "don't design it for me" but I find it easier to understand if drawn out.

One of the nice things about building simple MPU circuits these days compared with the 1980's is that, as westfw says it's easy to get hold of "large" RAM and ROM chips easily - it's much easier to provide 64K as 2 32k x 8bit SRAM chips than it is to interface to DRAM when you're just starting out (that said I'd probably go with a couple of 1Mx4 DRAMS if I wanted "large" amounts of RAM on a Z80 based system).

For small microcontroller style systems the 64kbit (8k x 8-bit) EPROMS and SRAMS are a good starting point. As I said use a 74LS138 to create a set of 8 select signals. These will then correspond to addresses in the range 0x0000-0x1FFF, 0x2000-0x3FFF and so on.

Thus to build a system with ROM at 0x0000-0x1FFF and RAM at 0x2000-0x2FFF the circuit looks like this





When reset the Z80 starts fetching instructions from address 0 so you normally need ROM there, typically you put a jump at address 0x0000 to the real initialisation code because in practice you only have 8 bytes to play with because of the RST instruction vectors.

You can expand for more memory, either ROM or RAM as needed by using the unused 74LS138 outputs. If 8k ROM and 8k RAM are sufficient you can keep the component count down by using memory-mapped I/O (though I'm not sure whether the standard Z80 peripheral chips support this, in fact I have a feeling that they don't).

« Last Edit: July 27, 2013, 05:16:36 pm by grumpydoc »

Logged

The following users thanked this post: TomS\_

**grumpydoc**

Super Contributor



Posts: 2705

Country:

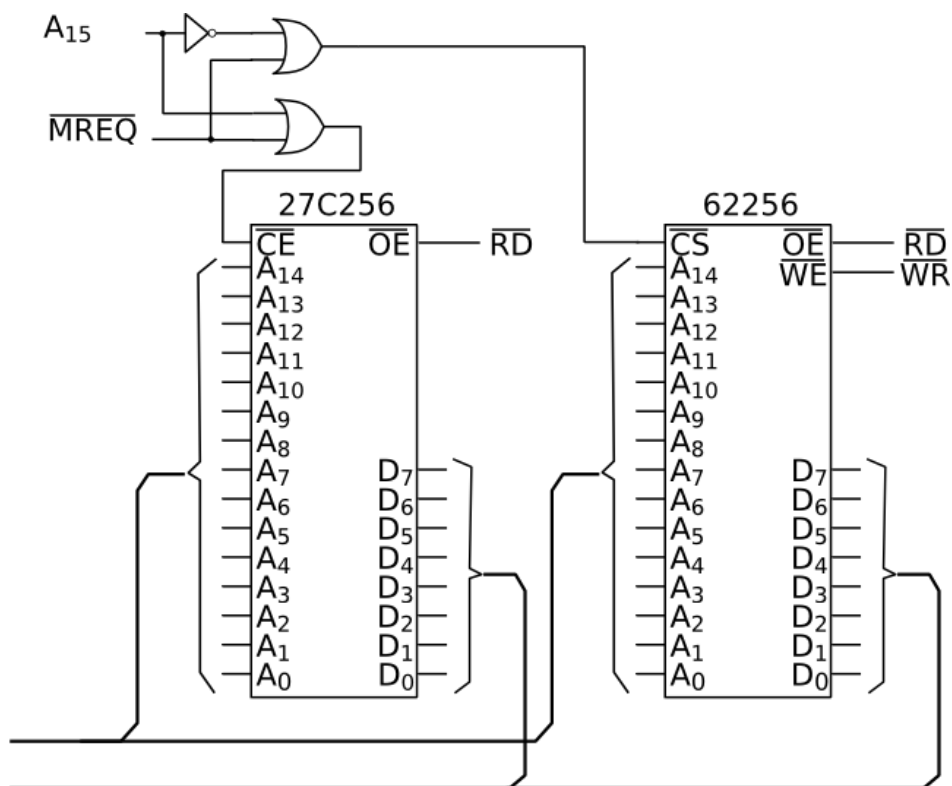


**Simple Z80 memory interfacing pt**

**2**

« Reply #12 on: July 27, 2013, 03:19:07 pm »

Carrying on from my last post if you want 32k ROM and 32k RAM it's almost easier - you can dispense with the 74LS138 and just use an inverter and a pair of OR gates to generate the chip select signals, like this



The ROM occupies the first 32k (0x0000-0x7FFF) and the ram the 2nd 32k (0x8000-0xFFFF)

You did mention that you had a 32k x 8-bit EPROM and 8k x 8-bit SRAM chips - in that case you could just use the circuit above but only connect A<sub>0</sub> to A<sub>12</sub> to the RAM chip. The result will be that the RAM contents repeat every 8k up the memory map from 0x8000 to 0xFFFF. It doesn't actually matter as long as you remember there's only really 8k of RAM and this "lazy" address decoding is very typical of embedded systems built around 8-bit microprocessors.

In fact if you wire a RAM socket up for a 62256 you can plug a 6264 into the socket and it will work. However note that because A<sub>13</sub> will now be connected to CS2 the RAM will only appear at addresses where A<sub>13</sub> is high - i.e 0xA000 to 0xBFFF and with a 2nd "mirror" at 0xE000 to 0xFFFF

If you wish to use the two 8k x 8 SRAMS then you can run A13 to CS2 on one and via an inverter to CS2 on the other. You will then have 16k of RAM addressed at 0x8000-0xBFFF with a mirror image from 0xC000-0xFFFF

« Last Edit: July 27, 2013, 08:56:38 pm by grumpydoc »

Logged

The following users thanked this post: TomS\_

grumpydoc

Super Contributor



Posts: 2705

Country:



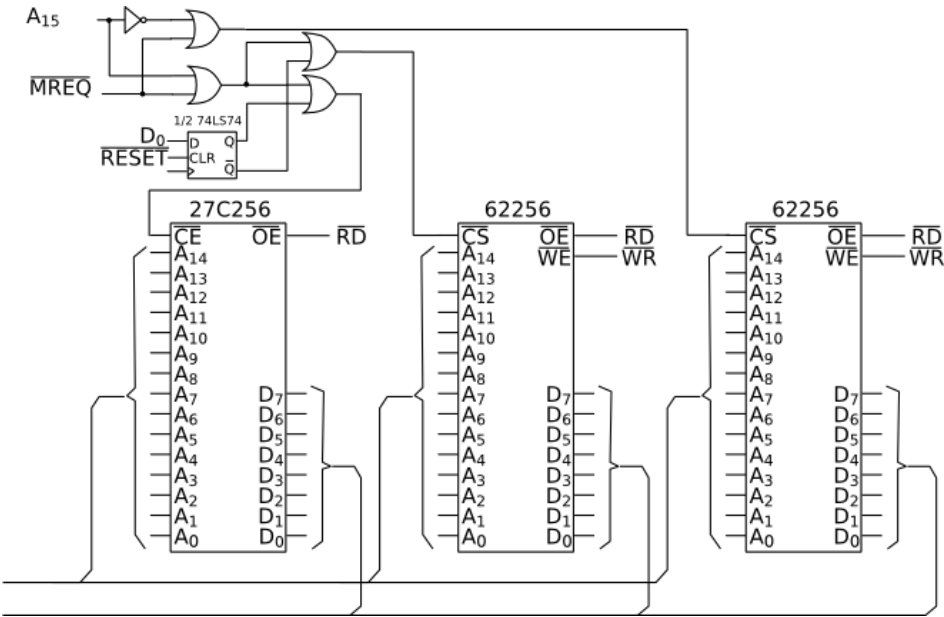
Simple Z80 memory interfacing pt 3

« Reply #13 on: July 27, 2013, 04:27:49 pm »

OK, so the previous two examples are fine for micro-controller projects but the OP was interested in CP/M. For that you need RAM at low addresses because the TPA (i.e where programs are loaded) starts at 0x0100. Also you really need 64k of RAM if at all possible.

This all provides a bit of a problem, how do we have ROM at address 0 for the initial cold reset and RAM at address 0 when running CP/M? I should point out BTW that for the moment I'm thinking of CP/M 2.2 and below - CP/M 3.0 supported bank switching but it wasn't around the last time I built a machine to run CP/M so I need to go and have a look at how it works.

We need to introduce some ability to switch one bank between ROM & RAM - the following (totally from memory so untested) circuit should do the trick



We now have two 32kx8 SRAM chips plus some additional gating with half a 74LS74 and a couple more OR gates. On reset the flip flop is cleared so the output will be low, this allows the chip select signal to pass to the EPROM. If a "1" is written to the flip-flop then the chip select will pass to the SRAM.

So, after a reset code in the ROM will run - as part of its initialisation it needs to copy some code up into addresses above 0x8000 and then jump to that code - the flip flop can then be switched to allow the whole 64k to be RAM.

I haven't shown how the clock input to the flip-flop is generated, so that can be considered an exercise for the reader 😊 but it needs to be the result of combining a decoded I/O address and the processor WR signal. I'll try to find time for some posts covering I/O interfacing in the next day or two.

Hope these posts have been helpful.

« Last Edit: July 27, 2013, 04:31:09 pm by grumpydoc »

Logged

The following users thanked this post: TomS\_

MrAureliusR

Supporter



Posts: 366

Country: 🇨🇦

frozenelectronics.ca



Re: Z80 single board memory bank switching

« Reply #14 on: July 28, 2013, 01:26:49 am »

Oh man, this is great information!!

I'm going to print out the whole thread (or at least PDF it), it's so helpful to see examples with my size of chips in the images! Thank you a million times grumpydoc! (Or as we say in Norwegian, tusen takk - a thousand thanks!)

I'm going to try and implement this right now. In the assembly notes for the Z80 monitor, this is how he has the memory mapped:

```
Code: [Select]
; +-----+
; ! $FB6C !    12 byte string buffer
; ! --- !
; ! $FB61 !
; +-----+
; ! $FB60 !    Buffers for various routines
; ! --- !
```

```

; ! $FB4D !
; +-----+
; ! $FB4C ! Cold/warm start control (1 byte)
; +-----+
; ! $FBBD ! Stack
; ! ... !
; ! $8000 ! Begin of RAM
; +-----+
; ! $7FFF ! ROM area
; ! --- ! RST $08 calls a system routine

```


Now, I'm eventually going to need to change some of that around, as I noted he implemented IDE (hence the FAT control block, etc). I'm spending a lot of time reading and re-reading the assembly and it's actually quite simple. I like how everything is laid out in assembly, much simpler than even C++ (which is where my experience is). The good part is that his memory map is split up exactly as you have example #2 split up, 32k of RAM and 32k of ROM.

If you're at all interested, I'm attaching the z80mon.asm file to show what I'm working with. First, of course, I've got to get the hardware all set up. So it sounds like I should just use my SIO for communication. The videos on putting together a Z80 that I've seen used a UART and MAX232 -- I could do either as I have all the proper chips. As for your examples on memory mapping, I also have a bunch of 74LS74's so that won't be a problem at all. (Plus a bunch of plain logic gates -- 4011, 7400, 7402, 7404, 7408, etc etc. Plus a bunch of the more obscure ones as well... like a 74S412 which I haven't been able to find a use for yet.)

Anyway, this is all going to take a bit of brain processing time in order for me to get it all straight. I'll report back as I progress!

EDIT: So I just spent the last hour completely removing all address and data lines and then setting up your example #2, with the single inverter and double OR gate. I've made sure all the lines are plugged in correctly, and everything seems to be working. (note: the Toshiba RAM chips only go up to A12, although I suppose this makes sense). However, until I can interface with my serial port on my computer, I can't know for sure. So I guess the next step is to get the SIO up and running. The datasheet on the SIO says that the Rx and Tx lines on the SIO output and receive at TTL voltage levels, so I do need to use a MAX232 with it.

I'm taking a break for a few minutes to let my mind wander in between bursts of working. Seems to be thankful for the rest 😊

 [z80mon.asm](#) (104.68 kB - downloaded 335 times.)

« Last Edit: July 28, 2013, 02:58:09 am by MrAureliusR »

 Logged

-----  
 Amateur Radio operator VA3XMR  
 -----

 **amyk**

Super Contributor



Posts: 6763



### Re: Z80 single board memory bank switching

« Reply #15 on: July 28, 2013, 10:23:28 am »

Another way you can get around the reset vector being at the bottom of memory and still have ROM at the top is to make the first 3 bytes read (from any address) be e.g. C3, F0, FF which will make it execute a JP FFF0 and then start executing there thereafter.

(The x86 series all start 16 bytes from the end of the address space, presumably due to this issue and a preference for ROM at the top of memory.)

 Logged

 **komet**



### Re: Z80 single board memory bank switching

Supporter



Posts: 155

Country:

Shenzhen Retroencabulator  
Mfg. Co.« **Reply #16 on:** July 28, 2013, 10:34:06 am »**Quote from: amyk on July 28, 2013, 10:23:28 am**

Another way you can get around the reset vector being at the bottom of memory and still have ROM at the top is to make the first 3 bytes read (from any address) be e.g. C3, F0, FF which will make it execute a JP FFF0 and then start executing there thereafter.

(The x86 series all start 16 bytes from the end of the address space, presumably due to this issue and a preference for ROM at the top of memory.)

It's easier to inject C3, C3, C3 into the data readout and start the ROM program at C3C3.

Logged

☐ **MasterOfNone**

Regular Contributor



Posts: 123

**Re: Z80 single board memory bank switching**
« **Reply #17 on:** July 28, 2013, 10:42:58 am »

Sorry amyk and komet, I don't understand, how can you easily fix the first three bytes read without a ROM at location 0x0000? Where are these values held?

Logged

☐ **amyk**

Super Contributor



Posts: 6763

**Re: Z80 single board memory bank switching**
« **Reply #18 on:** July 28, 2013, 10:58:01 am »

You can use a few logic chips to switch the value onto the data bus for the first 3 fetch cycles.

It might suffice to just use pullup/pulldown resistors and tristate the databus buffer if you're going for C3 C3 C3...

Logged

☐ **grumpydoc**

Super Contributor



Posts: 2705

Country:

**Re: Z80 single board memory bank switching**
« **Reply #19 on:** July 28, 2013, 10:58:50 am »

@MasterOfNone

**Quote**

Sorry amyk and komet, I don't understand, how can you easily fix the first three bytes read without a ROM at location 0x0000? Where are these values held?

They're not - you fake it by stuffing the values onto the data bus. TBH it's much easier to overlay ROM and RAM at the same address as I described.

One of the interrupt modes (mode1, IIRC) allows you to place any single instruction on the bus in similar fashion - I've never done that either, too much like hard work!

@MrAureliusR

**Quote**

(note: the Toshiba RAM chips only go up to A12, although I suppose this makes sense).

Yes as they're  $8k \times 8k = 8192 = 2^{13}$  hence 13 address lines  $A_0$  to  $A_{12}$ .

I had a 62256 in my 2nd diagram.

Logged

 **MasterOfNone**

Regular Contributor



Posts: 123



## Re: Z80 single board memory bank switching

« **Reply #20 on:** July 28, 2013, 11:18:02 am »

**Quote from: amyk on July 28, 2013, 10:58:01 am**

You can use a few logic chips to switch the value onto the data bus for the first 3 fetch cycles.

It might suffice to just use pullup/pulldown resistors and tristate the databus buffer if you're going for C3 C3 C3...

**Quote from: grumpydoc on July 28, 2013, 10:58:50 am**

@MasterOfNone  
**Quote**

Sorry amyk and komet, I don't understand, how can you easily fix the first three bytes read without a ROM at location 0x0000? Where are these values held?

They're not - you fake it by stuffing the values onto the data bus. TBH it's much easier to overlay ROM and RAM at the same address as I described.

One of the interrupt modes (mode1, IIRC) allows you to place any single instruction on the bus in similar fashion - I've never done that either, too much like hard work!

@MrAureliusR  
**Quote**

(note: the Toshiba RAM chips only go up to A12, although I suppose this makes sense).

Yes as they're 8k x8 8k = 8192 =  $2^{13}$  hence 13 address lines  $A_0$  to  $A_{12}$ .

I had a 62256 in my 2nd diagram.

Nope, sorry still don't get it. I'm missing something here because I can't see how you would have to fully decode the address so that the three bytes isn't repeated in the ROM/RAM address space.

I don't understand placing values on the bus, where is it latched? Do you mean writing to RAM?

 Logged

 **amyk**

Super Contributor



Posts: 6763



## Re: Z80 single board memory bank switching

« **Reply #21 on:** July 28, 2013, 12:15:36 pm »

Ignore the address bus, it's irrelevant for this.

You can use a multiplexer or similar to switch between the system data bus and a hardwired value C3. Its select input goes to the output of 2 flip-flops connected serially and clocked by the falling edge of /M1 with a 1 input. Reset loads them with 0. The multiplexer will force C3 on the data bus inputs of the processor until the 2nd M cycle, where it switches to the regular data bus and then things continue normally.


 Logged

 **grumpydoc**

Super Contributor



Posts: 2705

Country: 



## Re: Z80 single board memory bank switching

« **Reply #22 on:** July 28, 2013, 12:29:45 pm »

**Quote**

Nope, sorry still don't get it.

Did amyk's post help?

## Quote

I'm missing something here because I can't see how you won't have to fully decode the address so that the three bytes aren't repeated in the ROM/RAM address space.  
I don't understand placing values on the bus, where is it latched? Do you mean writing to RAM?

The suggestion is to **override** whatever data would be fetched from address 0x0000, 0x0001 and 0x0002 in the first three reads following a reset. One idea is that by arranging the correct pattern of pull-up (and pull down) resistors on the data bus it will read 0xC3 if nothing is driving the bus actively, however there are other ways such as amyk's suggestion of a multiplexor.

You then use a flip-flop to count "M1" cycles - following a reset, until the falling edge of the 2nd M1. Up to that point you disable ordinary memory access so the processor reads 0xC3, 0xC3, 0xC3. The first read will be interpreted as an opcode - 0xC3 is an unconditional jump. The 2nd two reads are the 16bit target address so the processor will execute a jump to address 0XC3C3. Assuming you have ROM at that address you can place your reset code there.

As it's always 0xC3 you don't need to bother decoding the address bus.

« Last Edit: July 28, 2013, 12:35:30 pm by grumpydoc »

 Logged

 **grumpydoc**

Super Contributor



Posts: 2705

Country: 



### Simple Z80 I/O interfacing pt 1

« Reply #23 on: July 28, 2013, 12:48:04

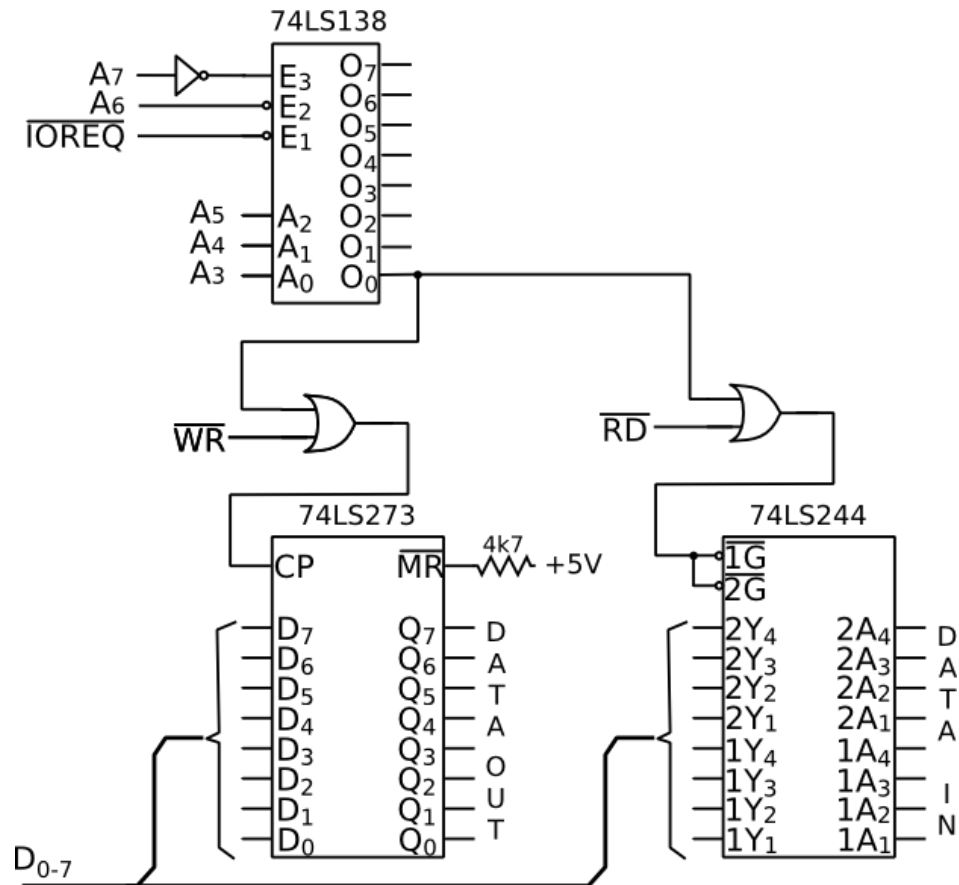
pm »

Following on from the "how to drive memory" posts a useful system needs some I/O. The hardware interface is similar to memory but officially only 8 bits of the address bus is used to give 256 I/O (or "port") addresses.

The simplest port just consists of a flip-flop (or latch) for output and a tri-state buffer (for input). It's worth adding at least one simple output port and connecting it to a LED (or several) for diagnostics. Although the Z80 has the dedicated PIO chip it needs to be correctly set up in software before it will do anything terribly useful.

Again we can use the 74LS138 as a decoder to produce 8 active low select lines. In the following circuit the first 64 port addresses are decoded in blocks of 8, so the first select is low for any access in the range 0x00 to 0x07, the second for accesses in the range 0x08 to 0x0F etc.





It would be possible to further sub-divide this down with another 74LS138 to give 8 signals corresponding to port 0x01, 0x02 etc but I haven't done this in the example - again this sort of incomplete decoding is very common in small embedded systems to reduce component count and costs.

« Last Edit: July 28, 2013, 01:16:16 pm by grumpydoc »

Logged

**MasterOfNone**

Regular Contributor



Posts: 123



**Re: Z80 single board memory bank switching**

« Reply #24 on: July 28, 2013, 01:04:43 pm »

**Quote from: amyk on July 28, 2013, 12:15:36 pm**

Ignore the address bus, it's irrelevant for this.

You can use a multiplexer or similar to switch between the system data bus and a hardwired value C3. Its select input goes to the output of 2 flip-flops connected serially and clocked by the falling edge of /M1 with a 1 input. Reset loads them with 0. The multiplexer will force C3 on the data bus inputs of the processor until the 2nd M cycle, where it switches to the regular data bus and then things continue normally.

**Quote from: grumpydoc on July 28, 2013, 12:29:45 pm**

Did amyk's post help?

Nice, I've never seen that technique used before but it looks good to me.

Logged

Pages: [1] 2 3 Next All **Go Up**

PRINT SEARCH

« previous next »

Share me



EEVblog Electronics Community Forum » Electronics » Beginners » Z80 single board memory bank switching

Jump to: => **Beginners** ▼ go



[EEVblog Main Site](#)

[EEVblog on Youtube](#)

[EEVblog on Twitter](#)

[EEVblog on Facebook](#)

[EEVblog on Library](#)

SMF 2.0.15 | SMF © 2017, Simple Machines

Simple Audio Video Embedder

SMFAds for Free Forums

[XHTML](#) [RSS](#) [Mobile](#) [WAP2](#)