The ZBC Project

The Zero Board Computer Project

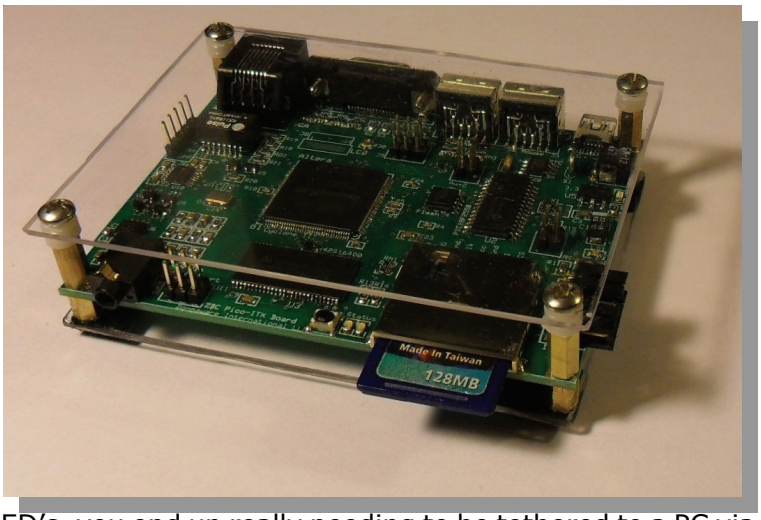
This article discusses the development and technical details of the Zero Board Computer. This is not intended to be a user manual as this is a hobby project and not a product.

Donnaware International LLP
August 30, 2010

Table of Contents

INTRODUCTION

ZBC is a really fun little PC-XT SoC (System on a Chip) that was is implemented on a Single Board circuit board shown in the photo. The board is called ZBC for Zero Board Computer instead of Single Board Computer just to be funny (in computers we always start at Zero not one right ??? ☺). All joking aside, ZBC is intended to be fun and educational and able to run DOS and perhaps general enough to run some other tiny OS (perhaps tiny Linux) or an older version of Windows (maybe Windows 3 or whatever). The design and code is based heavily on the Terasic (www.terasic.com) boards and the FPGA code is based heavily on the ZET computer (http://zet.aluzina.org) which runs on the Terasic DE1.



Motivation: The motivation for this project is that when, as a hobbyist, you decide to start getting into FPGA design, the first thing that you run into is to make anything "stand alone" can be problematic, unless the user interface is extremely simplistic. If the user interface is anything beyond a few buttons, switches and LED's, you end up really needing to be tethered to a PC via a USB cable. The other way to do it is to instantiate a PC on the FPGA itself, then you FPGA project can use simple PC tools like QBasic, Turbo-C or whatever to create your user interface. That is the idea behind ZBC. It is a PC that lives in a portion of your FPGA with a little room left over for you FPGA project. The fancy terminology for this approach is called an embedded processor (I guess because the processor is embedded in the FPGA).

The follow are the primary design rationale for the ZBC design:

Target Audience: The target audience is the serious hobbyist; someone who has some level of expertise in soldering, computers and programming, but not necessarily a super-geek or professional. For a hobbyist, cost is always a major concern, and since this is just for fun, my thinking is that the total cost of the project; (assuming some limited scrounging for connectors and such) should be in the $100-ish price range.

Open Source: The ZBC implementation is intended to be completely open source, no copyrights or patents. This includes all the FPGA Verilog HDL code, the BIOS source, all the circuit diagrams, PCB layouts, everything. As such, the ZBC is not for sale, you can not order one from anyone or anywhere. You have to build it yourself. But after all, that is the fun part!!!

FPGA Selection: If you are unfamiliar with FPGA's, start by visiting www.fpga4fun (a really easy read), then come back to this article. I selected the Altera Cyclone III because I know it. There, I said it, no other reason. Based on this target FPGA, the system needs to fit into at most, a 144 pin count TQPF package. You can not go lower on the pin count and still have a fun board with enough extra pins and space to play around with. With higher pin count chips and you start getting into FBGA (Fine Ball Grid Array) and UBGA (Ultra-fine Ball Grid Array) packages, which are difficult for most hobbyists to solder and you usually need special equipment (those packages have their pins all hidden underneath them and you really need professional wave soldering equipment to solder those types of packages). Also, the higher pin

count devices cost quite a bit more.

I have successfully soldered 144Pin TQFP packages before with a small tip soldering iron; the technique involves soldering the pins, while not really concerning yourself with blobing the solder all over the place, then cleaning up the inevitable solder bridges with the solder wick, or what I use is a de-soldering gun which has a sort of solder vacuum cleaner built into it. There are a number of web tutorials on this subject if you Google around. I have even seen hobbyist who use toaster ovens as a sort of substitute wave soldering system. The main warning here is that you need to be adept at soldering small form factor surface mount packages to attempt this project.

FPGA Cost: The Altera chip that I think fits very well into the niche is the Cyclone III, there are 3 versions that are good candidates:

Part #	User Pins	LE's	Cost (US)
EP3C10E144C7	94	10,320	$23
EP3C16E144C7	84	15,408	$27
EP3C25E144C7	82	24,624	$59

I selected the **EP3C16E144C7** because you have just enough pins to provide all the popular peripherals but still enough LE's (Logic Elements) to have fun with. If you notice, the more pins you have, the fewer LE's you get, this is because each pin takes up significant chip space for the IO driver circuitry which then cuts into space that could be used for LE's, or at least, that is how it has been explained to me. Also, the C7 is the speed grade, the 7 supposedly will go up into the 300Mhz range which is way higher than anything you will ever need for hobby stuff.

PCB Considerations: The PCB needs to be inexpensive as well, it is best to keep it down to a 2 layer board if possible to keep the cost of the board down. This is another reason to stay away fro the UBGA type packages because with that density you need at least a 4 layer board. I decided to use the ExpressPCB (www.Expresspcb.com) because I have always gotten good service from them and I am used to their system. Also, their pricing is not bad for the hobbyist and they do small quantities with no complaining.

The final board form factor I settle one was similar to the Pico-ITX (10 cm x 7.2 cm) which is a nice small size. There are not a huge number of cases available for this form factor yet, however, that may change in the future. I felt it was important to try to use some sort of standard and at the same time to use the smallest form factor possible.

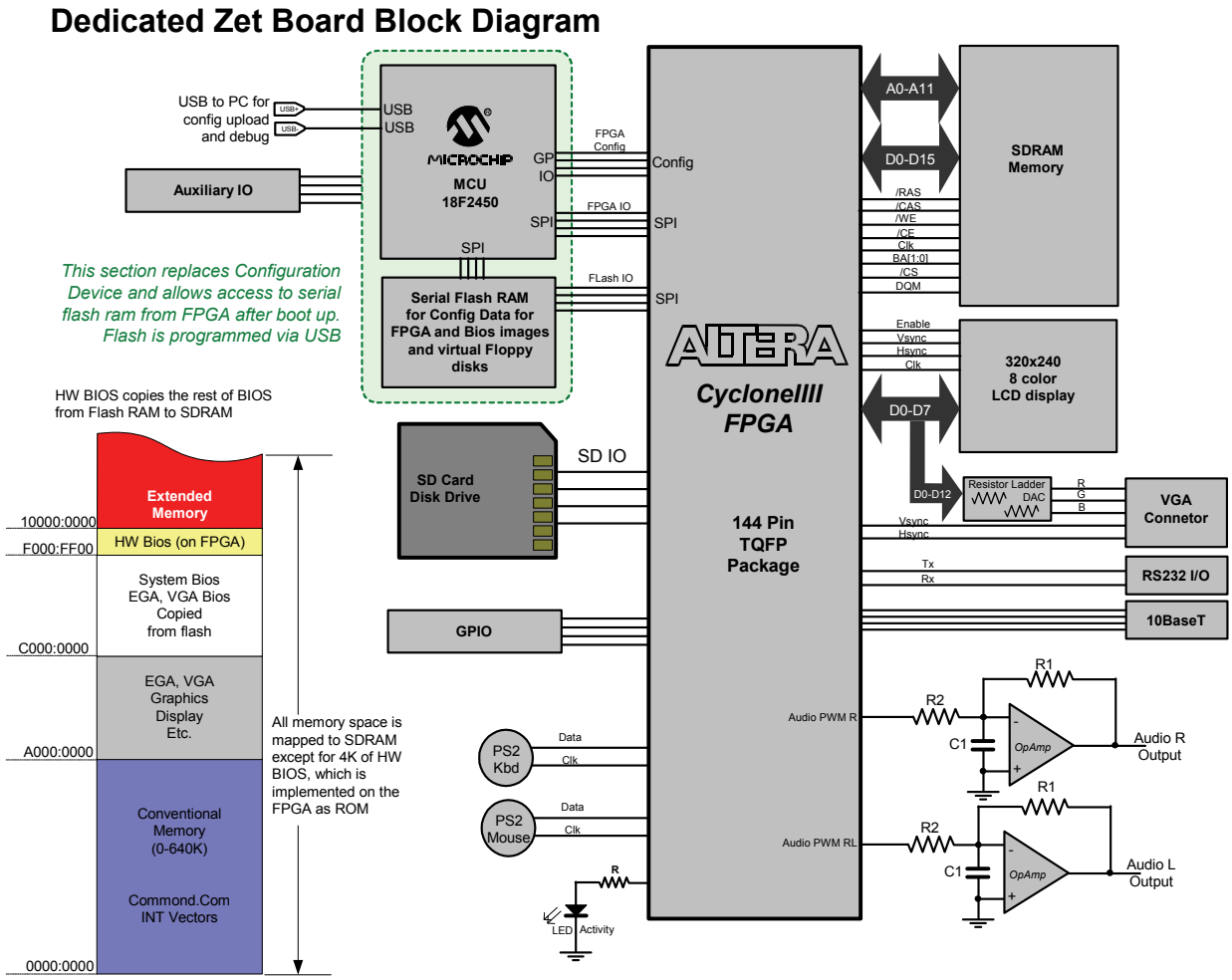
It is possible to reduce the size of the board further if you want to, the easiest way of doing that is to simple use header pins instead of on board connectors for the VGA connector and the 10BaseT connector. Also, if you move the SD Card Socket to the bottom of the board, a large amount of board real estate can be captured. It was not necessary to do either of these things in order to meet the Pico-ITX size, although I did opt to use a header pin row for the RS232 interface rather than placing a DB9 Connector on board.

HDL Consideration: The system should be such that someone can have fun with it without necessarily being a wiz at Quartus, or not need it at all [for those unfamiliar, HDL is Hardware Description Language, which is a special language engineers use to design chips. Quartus is the Altera design tool for doing this. Do not worry, ZBC is designed so you can have fun with it without knowing any of that, but it is more fun if you do, but you can get into that later if you wish].

Much of the HDL is taken directly from www.opencores.org and perhaps just tweaked here and there. Once you get ZBC running, the tweaking and hacking around is the fun part! ZBC come with a GPIO port built in so you can start reading and writing stuff to the IO pins from DOS. But the cool thing is to make up your own peripheral and drivers for something (see Appendix G for a tutorial on how to do that).

ZBC Block Diagram:

The block diagram for the ZBC design is shown below. Refer to this diagram in the subsequence discussion. There is more detail about the internal architecture of the FPGA Verilog HDL coding in Appendix F.



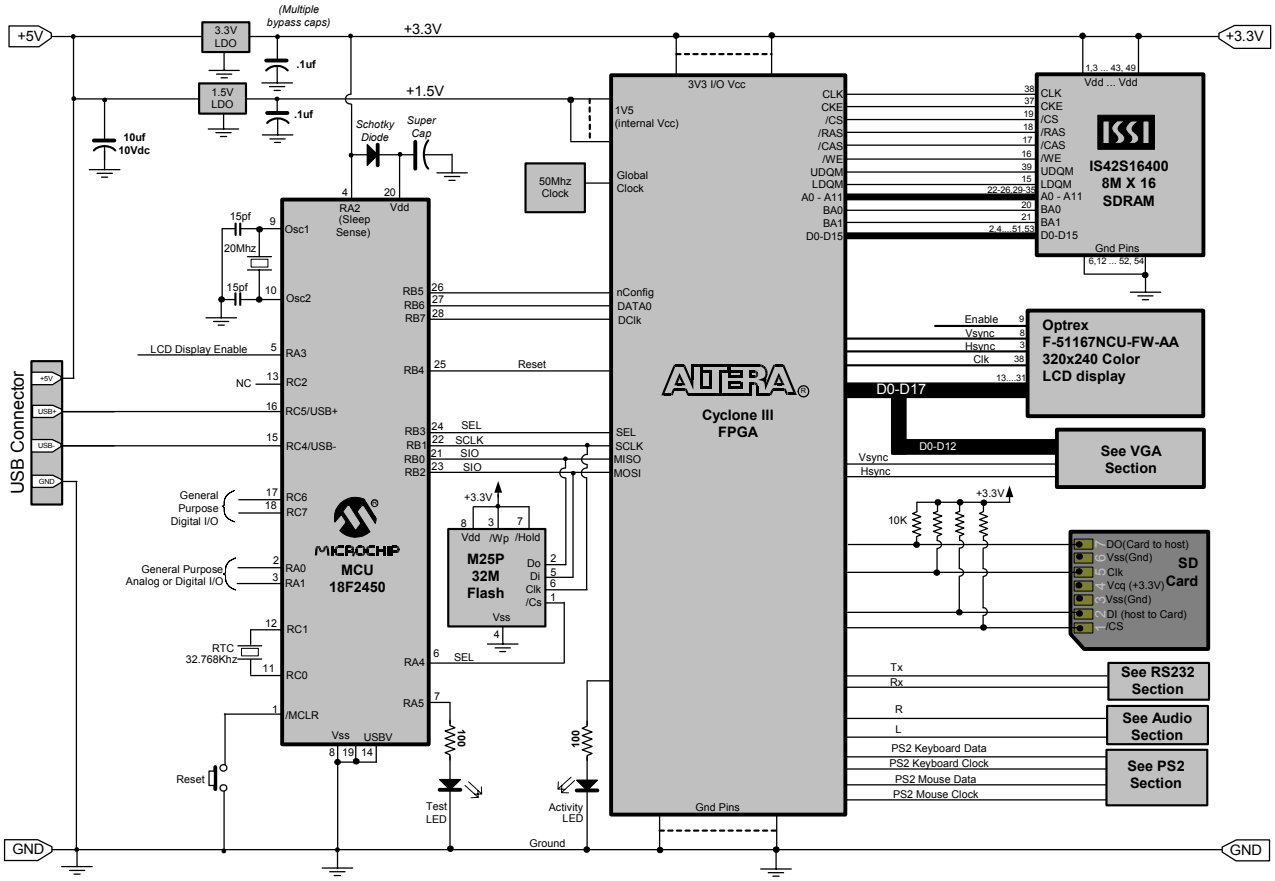
ZBC SYSTEM DESCRIPTION:

The following is a general description of the ZBC computer, the circuitry, layout and other various details.

FPGA Section: The detail diagram of the FPGA section is shown below. Notice that the FPGA chip is married to a PIC MCU. Typically the Altera chips will be configured using a dedicated configuration chip or the Altera USB Blaster (which you have to purchase from Altera

or a distributor). The USB Blaster is basically an FTDI USB chip and an Altera MAX PLD. Of course Altera wants to use their own products, hence the use of the MAX chip to interface via USB to a PC. However, using a Microcontroller, such as the PIC, provides far more flexibility and it also is much easier to implement more sophisticated functions (such as automatically selecting multiple configs and etc.) The PIC code is written in C Language and the USB stack is provided using an off the shelf library function. A user program residing on your PC interfaces to the PIC, which provides the option of uploading the FPGA config file directly into the FPGA or storing it (or multiple versions) on Flash RAM.

FPGA Section



BIOS BOOT LOADER: The key to keeping the design small and the pin count down, as discussed previously, is in the design of the BIOS boot loader and the Memory Map. The concept is that everything will run out of the SDRAM except for a small 256 byte section of BIOS that will reside at 0xF000:FF00 to 0xF000:0xFFFF (the very tippy toe top of the 8086 memory map. When the CPU boots, it jumps up there to the last memory location and you need to have a far jump in that spot to do the boot strapping to get things going. This little 256 byte section of bios will be mapped to ROM implemented using the M9K RAM on the FPGA (sort of built in ROM that you configure in the FPGA).

The Bios, is split into an assembly language portion and a C Language portion (listing included in this distribution). The more complicated functions are passed off to the C Module. This keeps the ASM module simpler so we don't have to be uber assembly language wizards to get things working. Also, the compiler and assembler used for the BIOS is the Openwatcom compiler and

assembler available for free at www.openwatcom.org

A small section of the ASM module is placed in the 256 byte block of on FPGA ROM. This section contains a small routine to load the main BIOS into SDRAM from a serial FLASH chip into the correct BIOS RAM locations. The benefit of this approach is that a small serial flash chip can be used to keep cost and pin count down while not sacrificing performance since the BIOS will be running in SDRAM.

The advantage of doing it this way is twofold; 1) things run faster in RAM than on a flash ROM, 2) by doing it this way you can use a serial flash RAM chip which keeps the pin count down, see next section.

FLASH RAM and FPGA Configuration: The second unique idea is to, instead of using a traditional configuration device, to use a standard serial flash chip such as the SST25VF064B 4MByte x 8 serial flash or similar (about $2 each). This device contains the load-able binary BIOS as well as the FPGA configuration file (.rbf) as well as a virtual floppy diskette. There are pin and software compatible versions of this chip available in larger sizes if you want to spend a little more money and get a larger capacity FLASH RAM on board. Not a big deal though, because remember we have the SD CARD slot and with that, we get tons-o space.

On power up, the MCU will assume SPI master control of the flash to access the RBF configuration file to configure the FPGA. The PIC reads the RBF file and uploads it into the FPGA chip. Once that is done, the FPGA kicks itself off and starts running.

Once the CPU is running on the FPGA, the MCU goes into SPI slave mode. So now the CPU becomes the SPI master for both the MCU and the serial flash. The BIOS then access the flash via an I/O port.

The MCU chip PCI18F2450 is also a very inexpensive chip (under $3) and most hobbyist have the ability to program them (PIC programmers are common and easy to get). This particular one has the USB interface built into it. This means the user can connect the board to his PC and down load new firmware, both FPGA configurations or BIOS without the use of or knowledge of Quartus and without needing a special FPGA programmer unit like the USB Blaster.

Since this is an open project, if you wish to later tinker around with it, then that is easy to do that as well. You just have to remember when you set up your Quartus project, to click on the .rbf check box to tell it to make that type of configuration file.

Another advantage of this approach is that the MCU has enough pins that a small set of DIP switches could be connected to the PIC MCU to tell it to boot up different versions of the CPU, BIOS, OS and you could also have a number of floppy images that could be switched in and out. It really just depends on the size of the flash chip you want to install.

Also, the PIC has a built in UART so you could also hook up a small character type LCD to it and send various debug messages to it. The PIC also has built in A/D converter than can be accessed via the SPI interface to the ZBC CPU.

Real Time Clock and CMOS Ram: A small DS1302 RTC is attached to the PIC MCU which provides the equivalent of the CMOS and RTC functions normally found on a PC. In other words, the PIC MCU looks to ZBC CPU like CMOS BIOS ram. Since the PIC is accessible via the

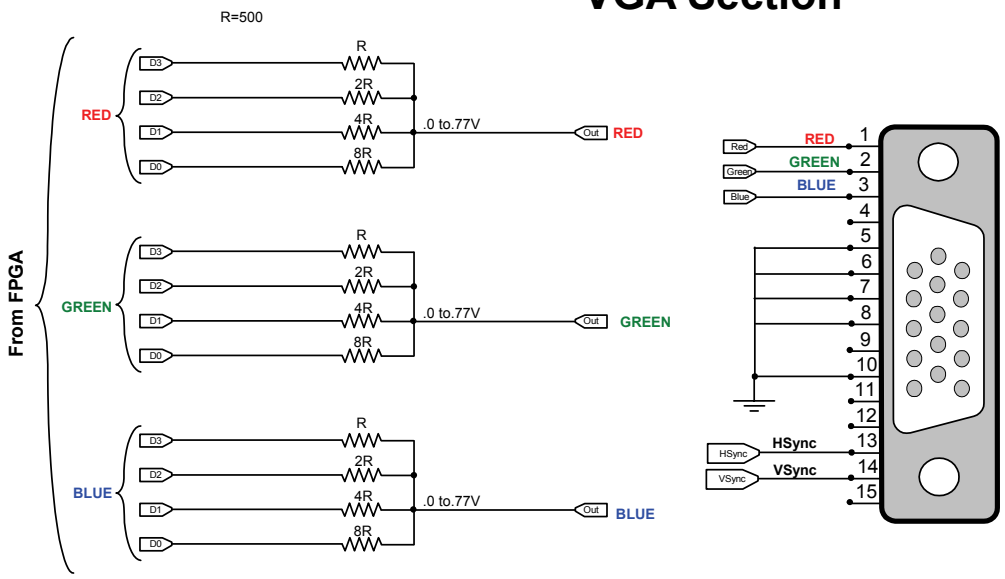
USB port from your PC, that means the equivalent of the BIOS set up is done by connecting your ZBC to your PC via the USB port and configuring it.

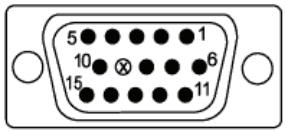
SDRAM: The SDRAM chip used is the ISSI IS42S16400 8M x 16 SDRAM which provides plenty of RAM for DOS applications including EMS, RAM drives and so on. There is nothing too fancy here, just a nice big ole memory chip.

VIDEO VGA/LCD: The VGA uses a simple resistor network to provide 4096 color (4 bits per color, RGB). The video ram is implemented on the SDRAM. The VGA and LCD interfaces share the same pins so it will not be possible in this implementation to use both at the same time. There will need to be a separate configuration for each.

Right now, the VGA buffer is implemented using FPGA RAM and has not been moved to SDRAM yet. Moving the video buffer would then make it possible to run graphics applications (Windows 3 for example) . Because of this limitation, at the present time, ZBC is only capable of running text mode applications.

VGA Section





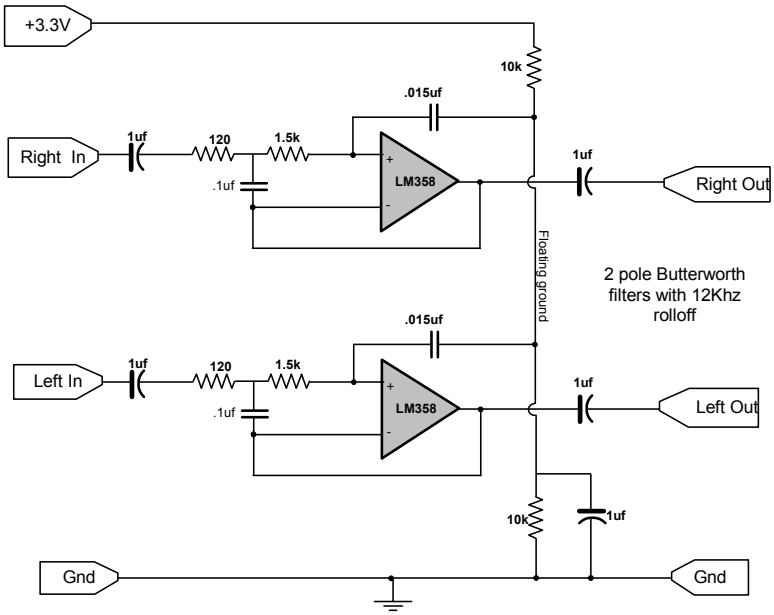
Female

Signal Name	Pin	Notes
RED video	1	Analog signal approx 0.7 volt, peak, 75 ohm
GREEN video	2	Analog signal approx 0.7 volt, peak, 75 ohm
BLUE video	3	Analog signal approx 0.7 volt, peak, 75 ohm
Monitor ID #2	4	(see note1)
Digital Ground	5	General "Ground" for the video system.
RED ground	6	\ Video signals should have a separate ground
GREEN ground	7	| path with pixel rates over 30MHz
BLUE ground	8	/ which is almost always
KEY	9	Not used
SYNC ground	10	TTL return for the SYNC lines.
Monitor ID #0	11	(see note1)
Monitor ID #1	12	(see note1)
Horizontal Sync	13	Digital levels (0 to 5 volts, TTL output)
Vertical Sync	14	Digital levels (0 to 5 volts, TTL output)
Not Connected	15	Not used

OPTIONAL LCD PANEL: A connector for an optional dedicated LCD panel is also provided. The panel is a surplus panel that is 320 x 240 8 color and the cost is $15 for the panel from Electronics Goldmine.

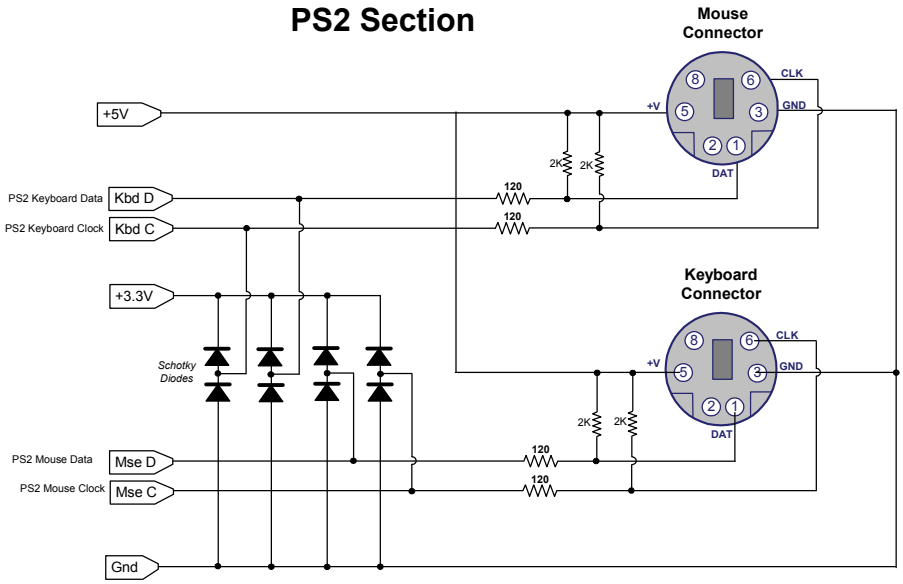
AUDIO: To keep the parts down and the design simple, I think that PWM DAC is the way to go. The audio is generated on the FPGA but sent out in PWM format so you only burn 2 pins. A simple low pass filter using a simple LM358 OP-Amp cleans up the signal. An LM386 could be added to boost the signal up to drive small speakers or a headset. A simple sound demo program is included at the ZBC Github you can download to see how to make this work.

Audio Section



PS2 Keyboard and Mouse: A simple level shifter is used to change from 5V to 3.3V (see Schematic below for detail).

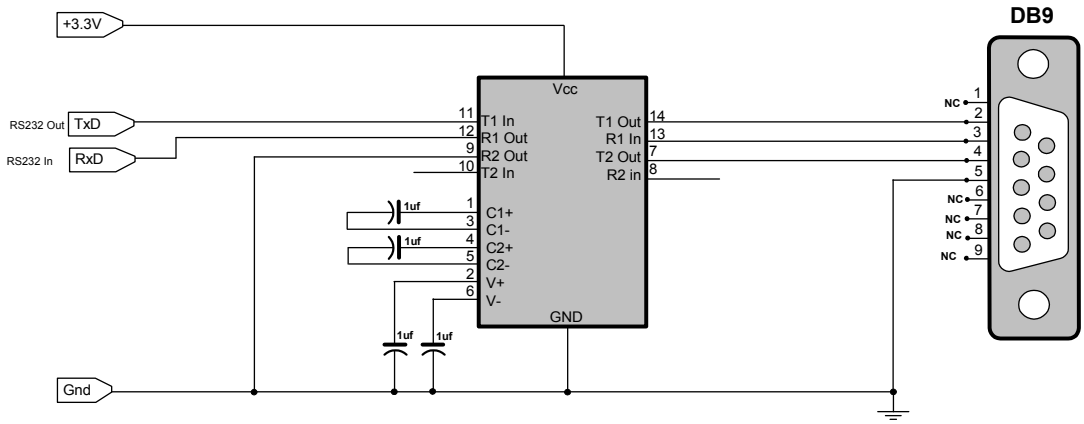
PS2 Section



The FPGA code for ZBC emulates the standard PC Keyboard and Mouse down to the scan code level so all the regular DOS programs work fine as far as I can tell.

RS232: A Basic Max232 circuit is used. It is set up such that you can use just Tx and Rx or you can sacrifice 2 GPIO pins and option for also having two hardware handshake pins (e.g. RTS, CTS). As mentioned earlier, the DB9 connector is not mounted on the PCB, instead a 5 pin DIN header is used and a simple pig tail with the DB9 on the end of that cable. For RS232, this is actually a better arrangement since it makes it easy to simple wire up the pig tail in whichever arrangement meets your needs; DCE or DTE.

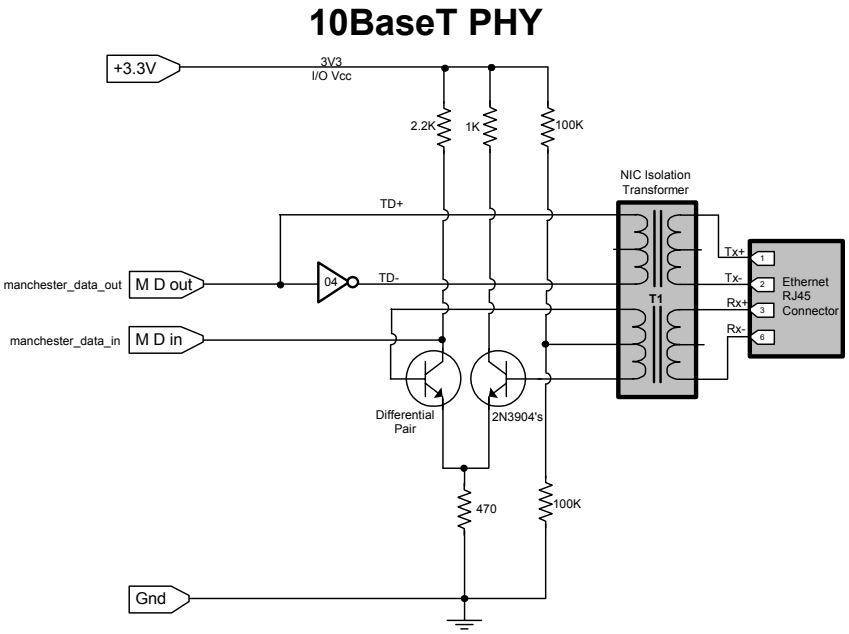
RS232 Section



SD CARD Based HARD DISK: The SD Card interface is very straight forward, it is simple a matter of connecting the /CS, DO, DI and CLK lines of the SD CARD to IO pins of the FPGA (see the FPGA section above for details. Since this is NOT a real IDE type hard drive, the BIOS had to be fiddled around with to get this to work. This is OK because just about all DOS programs go through the OS or Bios calls to access the drives. Basically, the DOS Int13 calls (disk IO) gets handed off to a C subroutine in the BIOS that translates from Head, Track, Sector to the correct SD CARD address locations.

Floppy Drive: Similar to the above, but the floppy image resides on the serial flash ram. The serial flash RAM I/O is very similar to the SD Card interface (some of the chips it is identical) hence, much of the SD Card code is re-used. The PIC MCU initialized the flash on bootstrapping so the code would be actually a simplified version of the SDIO.

10BaseT Interface: Below is the physical interface for the 10BaseT port. This is a simple circuit, the idea is to implement all the logic inside the FPGA to keep the part count and costs down. The transmit interface is just a single TTL buffer fed into the Ethernet Isolation transformer. The Receive interface is a simple transistor differential pair as shown below. The Transistors convert the pulses on the Ethernet line to Digital signals that are fed directly to the FPGA. A PHY/MAC core provides all the logic to drive this simple circuit.



Game Port and GPIO: Based on this design, there are a few pins left over which are connecter to headers pins on the board labeled Game Port and GPIO. The game port does not provide any interface circuitry for an actual game controller, that would have to be provided off board. It does provide Ground, +3.3V and 4 input pins, which is all you really need. Basically the game port is a simple 556 dual timer set up (see Appendix G for details on how to implement this). The GPIO header provides Ground, +3.3V and 2 input pins and 2 IO pins. Both the Game Port and GPIO pins can be accessed using the GPIO module of the ZBC CPU, but you can alter this by recompiling in Quartus.

Parts List and Cost break down:

The cost of the SBC would be approximately:

Part Number	**Cost (USD)**	**Description**
EP3C10E144C7	$22	16KLE Altera Cyclone III
IS42S16400	$ 3	8M x 16 SDRAM
PIC18F2450	$ 3	USB Microcontroller
SST25VF064B	$ 4	4Mx8 Flash RAM
PCB	$43	(depends on vendor)
Various Connectors	$0	Scrounging
50Mhz Oscillator	$2	Main FPGA Clock
LDO regulators	$4	FPGA Power Supply
Other miscellaneous	$0	Scrounging, and more Scrounging
Subtotal:	$81	(+ tax & shipping)

That leaves $20 for miscellaneous items and parts. Assuming you can scavenge some connectors and other pieces (an old PC motherboard would be a good source of connectors for example) I think the motivated hobbyist could build it for about $100 out of pocket.

FPGA Pin accounting: Below is an accounting of the pin assignments. Based on this count, any of the EP3C16E144C7 with 84 user pins fits perfectly leaving only 2 dedicated input pins unassigned.

Pin Name/Usage	Output	Input	Dir.	I/O Standard
Clock (Dedicated Clock Input)		1	input	3.3-V LVTTL
Reset Input		1	input	3.3-V LVTTL
GPIO	2	8	output	3.3-V LVTTL
SDRAM Addr[11:0]	12		output	3.3-V LVTTL
SDRAM BA[1:0]	2		output	3.3-V LVTTL
SDRAM /CE	1		output	3.3-V LVTTL
SDRAM Clk	1		output	3.3-V LVTTL
SDRAM Data[15:0]	16		bidir	3.3-V LVTTL
SDRAM DQM[1:0] (Permanently tie Low)			output	3.3-V LVTTL
SDRAM /CAS	1		output	3.3-V LVTTL
SDRAM /RAS	1		output	3.3-V LVTTL
SDRAM /WE	1		output	3.3-V LVTTL
SDRAM /CS	1		output	3.3-V LVTTL
SD_Card_MISO		1	input	3.3-V LVTTL
SD_Card_MOSI	1		output	3.3-V LVTTL
SD_Card_SClk	1		output	3.3-V LVTTL
SD_Card_SS	1		output	3.3-V LVTTL
SPI In		1	input	3.3-V LVTTL
SPI Out	1		output	3.3-V LVTTL
SPI Clk	1		output	3.3-V LVTTL
Flash SPI Sel	1		output	3.3-V LVTTL
MCU SPI Select	1		output	3.3-V LVTTL
LCD Vsync			output	3.3-V LVTTL
LCD XClk	1		output	3.3-V LVTTL
LCD Hsync			output	3.3-V LVTTL
LCD Data [7:0] (paralell /VGA)			output	3.3-V LVTTL
LCD Enable (tied to MCU)			output	3.3-V LVTTL
VGA Red [3:0]	4		output	3.3-V LVTTL
VGA RGreen [3:0] (paralell / LCD)	4		output	3.3-V LVTTL
VGA Blue [3:0] (paralell / LCD)	4		output	3.3-V LVTTL
VGA Hsync (paralell / LCD)	1		output	3.3-V LVTTL
VGA Vsync (paralell / LCD)	1		output	3.3-V LVTTL
PS2 Clk Kbd	1		bidir	3.3-V LVTTL
PS2 Data Kbd	1		bidir	3.3-V LVTTL
PS2 Clk Mouse	1		bidir	3.3-V LVTTL
PS2 Data Mouse	1		bidir	3.3-V LVTTL
Audio PWM L	1		output	3.3-V LVTTL
Audio PWM R	1		output	3.3-V LVTTL
10BaseT Tx	1		output	3.3-V LVTTL
10BaseT Rx		1	input	3.3-V LVTTL
RS232 UART RxD		1	input	3.3-V LVTTL
RS232 UART TxD	1		output	3.3-V LVTTL
Subtotal input and output pins	68	14		
Total pins	82			

Summary:

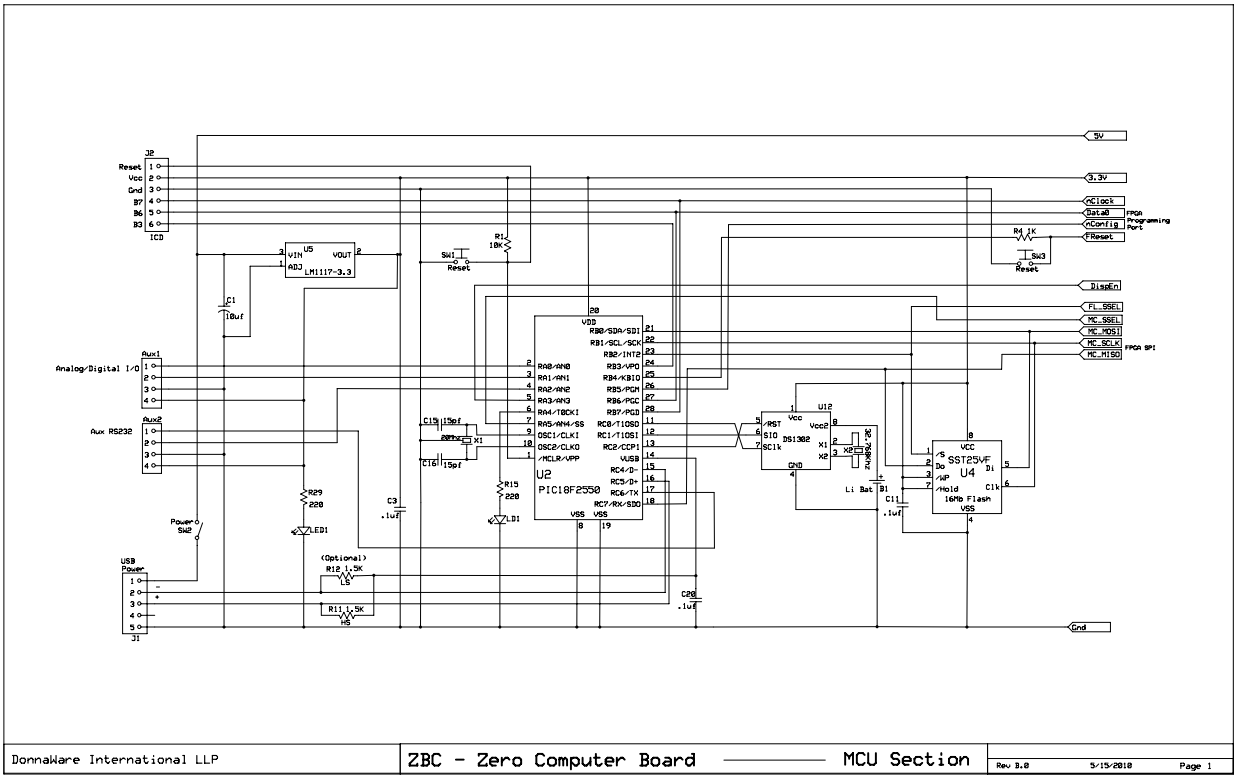
Well, that is the basic overview, to get more into it you really just have to download the sources and dig in. The rest of this document contains various appendices that contain further detail information such as detailed schematic diagrams, more photos, PCB layout and etc.

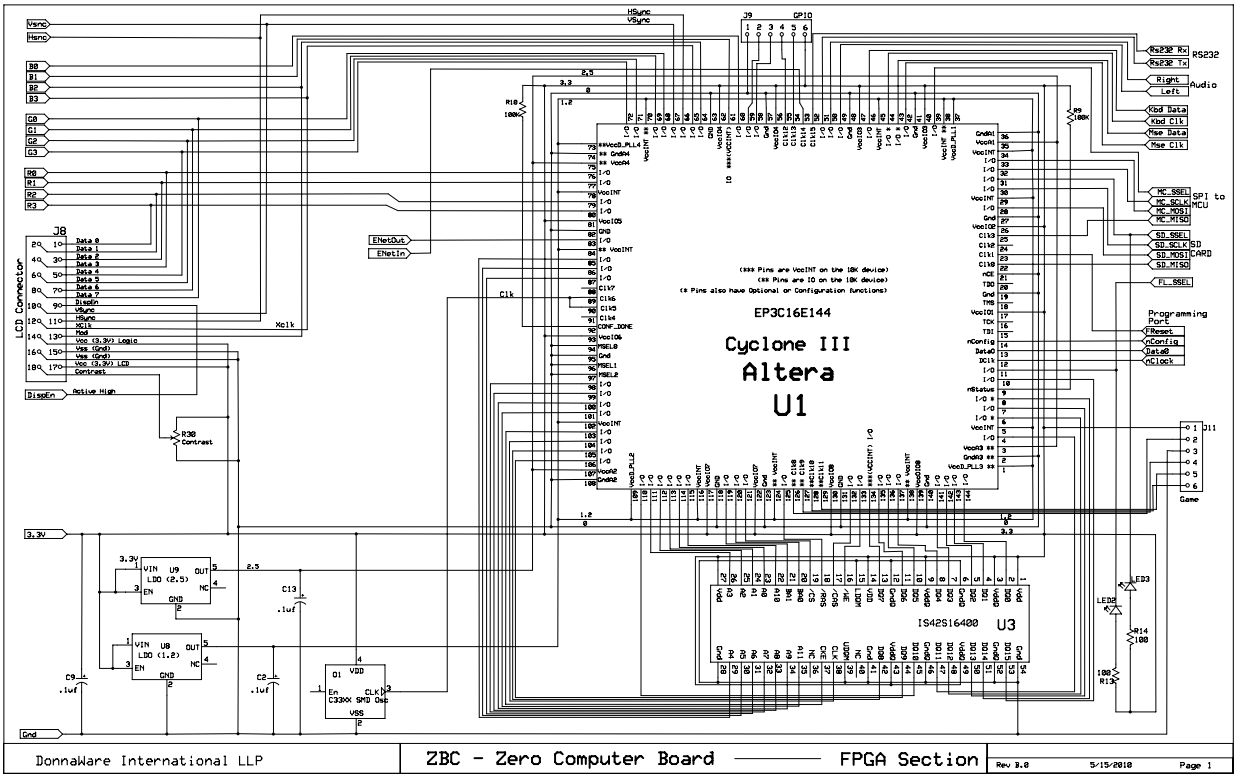
I think if you really look at the schematic of ZBC, you will have to agree that considering the functionality that is packed into it -- I mean it is an entire PX/XT with all the peripherals -- it has a really pretty surprisingly small component count and is really a pretty simple circuit design. This was on purpose, I really tried to keep the part count down and take maximum advantage of the FPGA and plow the functionality into it.

Where to find the sources: As mentioned at the beginning of this document, ZBC is completely open source. All the source files are available at my Github site: TBD

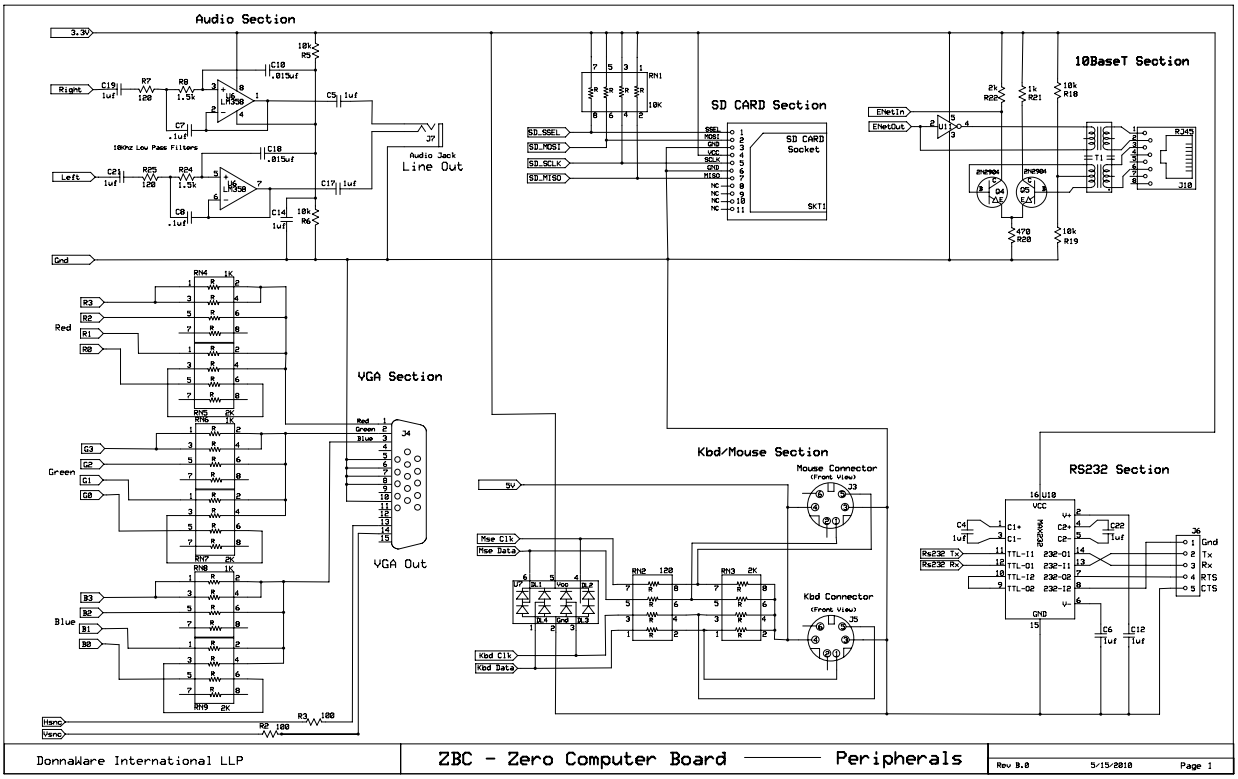
Good luck and enjoy !

Appendix A: Detailed Schematic diagrams:

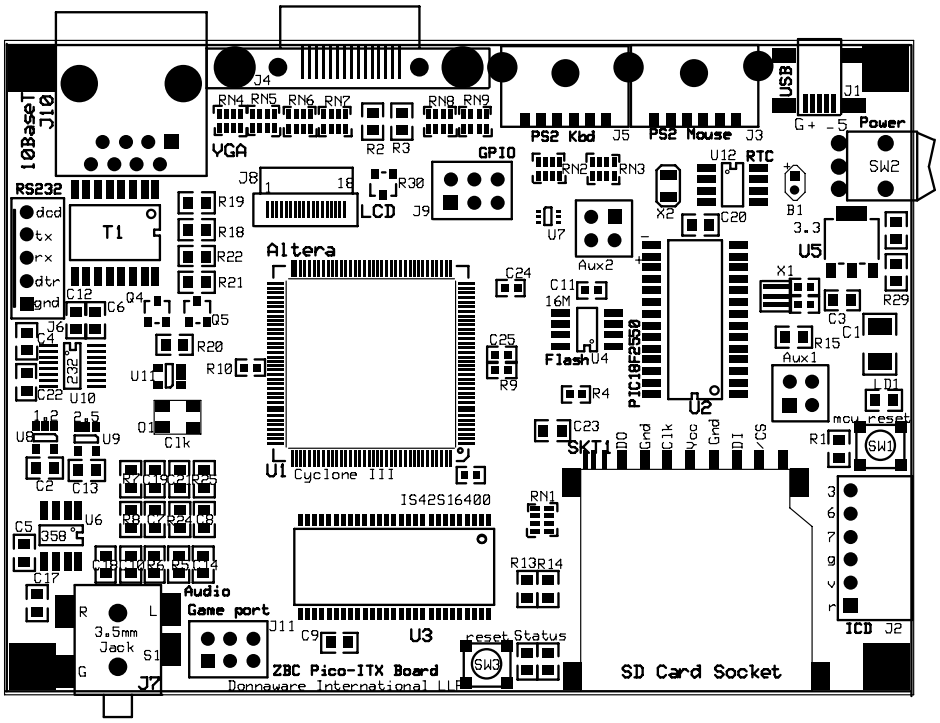


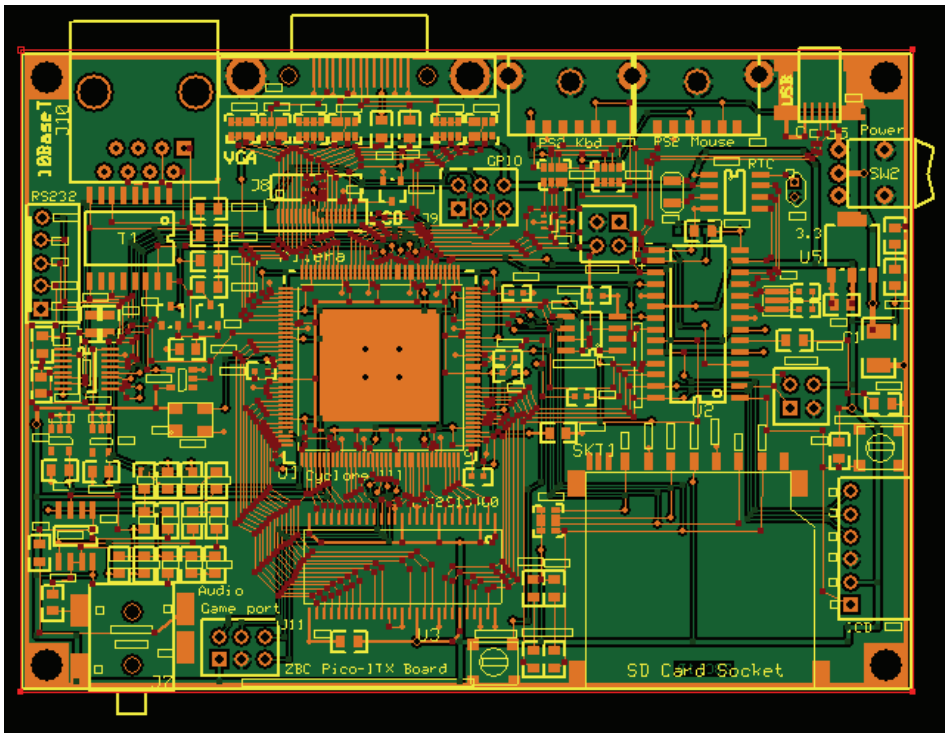


Appendix A: Detailed Schematic diagrams:

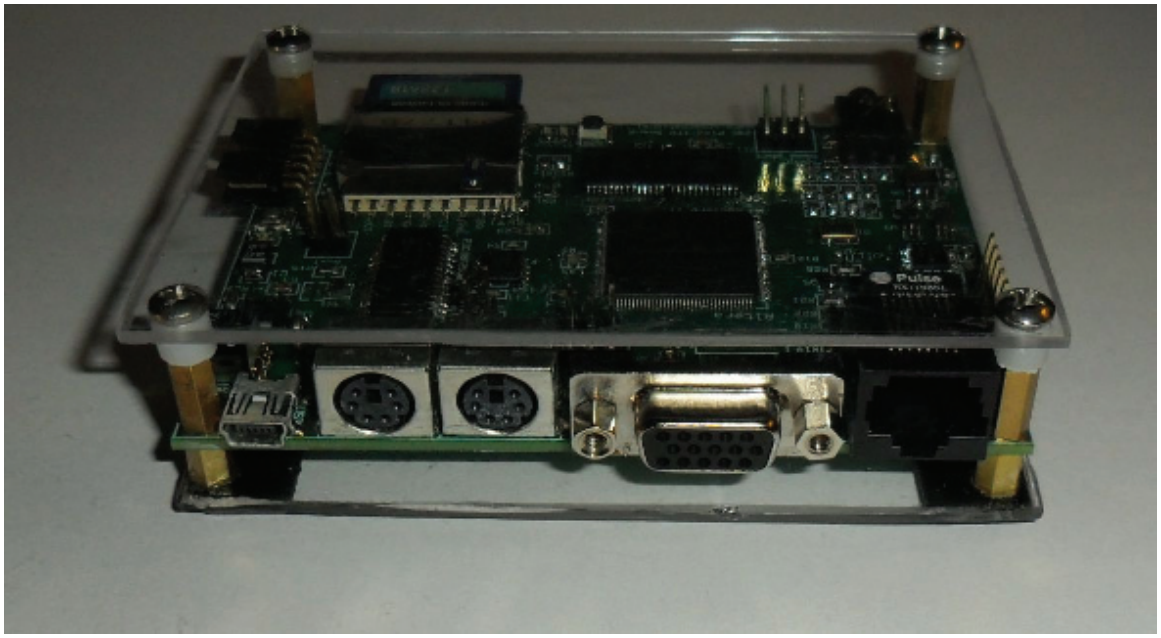


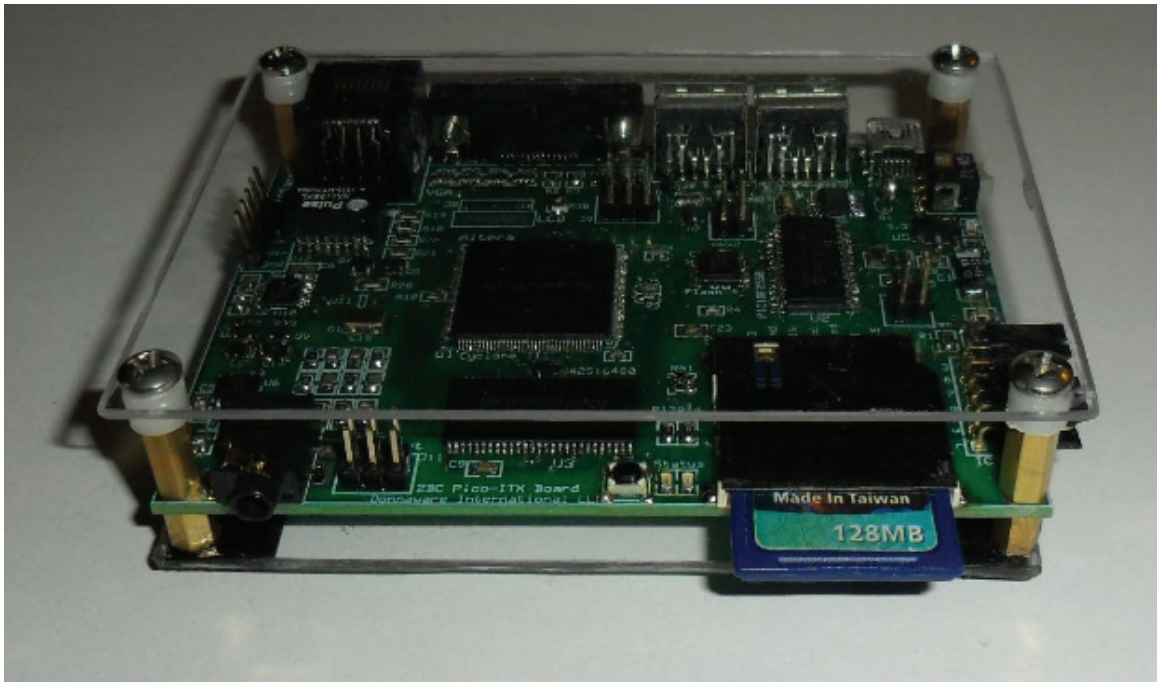
Appendix B: PCB Layouts:

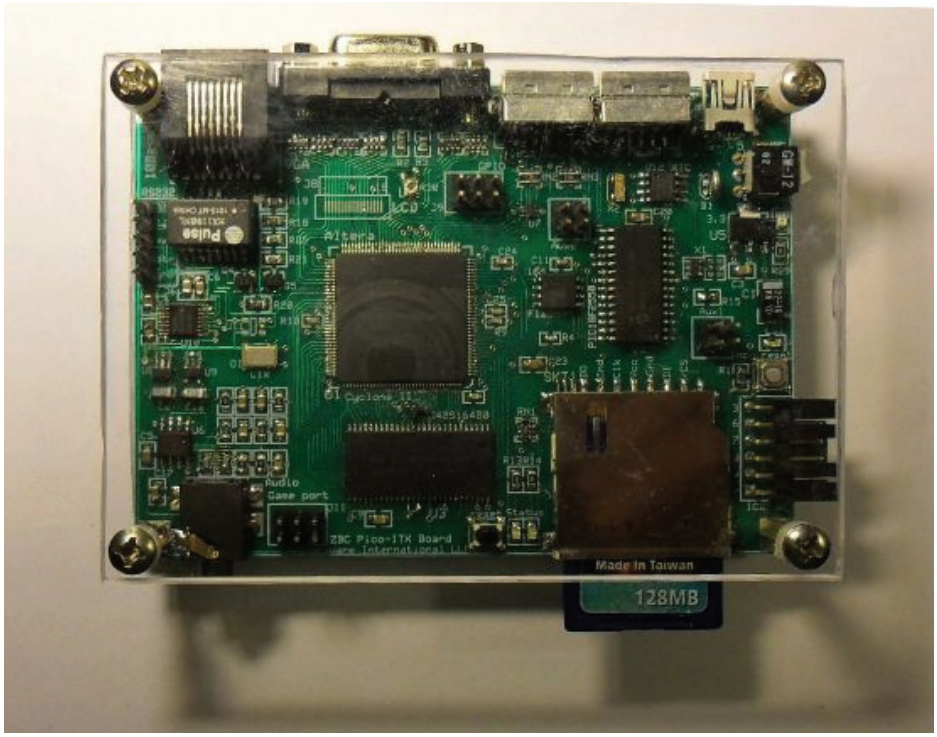




Appendix C: More Photos:






Made in Taiwan
128MB


F50L

Appendix D: Hints about Running DOS:

The first thing is that you need to get an older version of DOS. This is because the ZBC is an 8086 compatible CPU. I suppose that a 32 bit 386 compatible could be something that might come along someday, but for now, we are a 16 bit computer. So that means DOS 6.22 and earlier will run. Also, Windows 3.0 or earlier. If you do not happen to have those old DOS diskettes laying around, no problem, you can get copies of this software at this web site http://vetusware.com/category/OS . As it has been explained to me, these older versions of software are off copyright or basically no longer claimed by the owners. Also, the 8086 processor is off patent and such. So basically, this stuff is so old, we can play around with it and learn and not run afoul of the Central Scrutinizer.

On my Github site (url=tbd) I have a FreeDOS (www.freedos.org) super simple boot-able image that is ready to go you can use to get started. These image files, .img, are basically linear images of a disk. To create and manipulate these images, I highly recommend getting a copy of Winimage (www.winimage.com) loaded on your PC. You can make your own floppy image You can also make a hard drive image that is boot-able that can be loaded onto an SD CARD. Also, There is a utility for doing this on my Github under Tools.

ZBC can run most DOS programs, but there are some limitations. I have tried to make it as hardware compatible as possible but there are some things are harder to do. For example, it does not have a regular 8237 DMA controller nor does it have a true 8259 Priority Interrupt Controller (PIC). It has a simplified PIC that handles the Interrupts but does not have all the bells and whistles of the real one. So some programs that depend on those things will not work right. For one, the SoundBlaster depends on the 8237 DMA, so a lot of programs that use those types of drivers will not work right. I do have some example Turbo-C programs on Github that shows how to still play sounds.

Things that do work are the RS232 type programs and some games, Norton Commander, Turbo-C, QBasic and probably many others. You may have to just try things and see.

Below is the memory map and IO port assignments. Of course, if you have Quartus and know Verilog HDL, you can change all this around if you wish. Just check out the ZBC.v top level Verilog file.

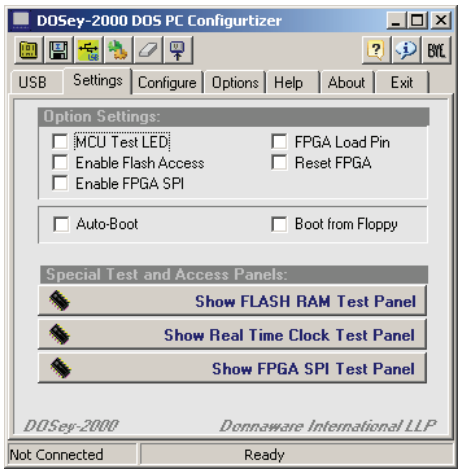
```
Memory and IO assignments:

Type    Range                Description
------  -------------------  ---------------------------------------
Memory  0x0_0000 - 0xF_FFFF  Base RAM, SDRAM range 1MB
Memory  0xF_FF00 - 0xF_FFFF  Bios BOOT ROM Memory
Memory  0xB_8000 - 0xB_FFFF  VGA Memory (text only)
IO        0xf100 -   0xf103  GPIO Registers
IO        0xf200 -   0xf20f  CSR Bridge Registers
IO        0xf300 -   0xf3ff  SDRAM Control Registers
IO          0x60 -     0x64  Keyboard / Mouse IO Registers
IO         0x100 -    0x101  SD Card IO Registers
IO        0x0238 -   0x023f  SPI Flash Registers
IO        0x03c0 -   0x03df  VGA IO Registers
IO        0x03f8 -   0x03ff  RS232 IO Registers
IO        0x0210 -   0x021F  Sound Registers
IO        0x0360 -   0x0368  10BaseT IO Registers
```

Appendix E: About the Configurator Program:

In order to set up ZBC, you need a program to communicate with your PC. After all, all these configuration files, floppy image files and etc. have to all get uploaded onto the Flash RAM via the PIC microprocessor over the USB line. To accomplish this easily, I have written a program that I call the DOSey Configurator. It is written in Borland C++ Builder and uses a special VCL component to take care of all the USB stuff. Of course the source and executable for this is all up on my Github.

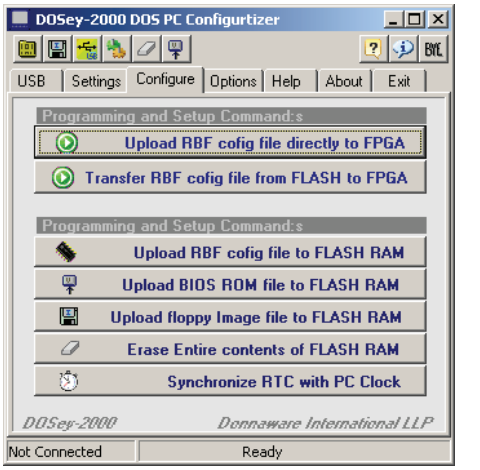
When you first start up **DOSey**, it looks like this screen print to the right. You can click on the big button in the middle to check to make sure that the USB Connection is working right between your PC and ZBC. It will show you a message telling you if it is or not.

Once you check that, you can see there are a number of tabs across the top; "Settings", "Configure", "Options", "Help", "About" and "Exit". I think most are pretty self explanatory but let's start with Settings.



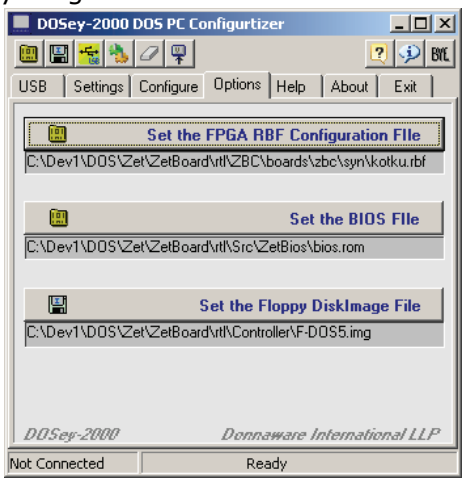
The **Settings** tab looks like this. There are a bunch of buttons and panels here. The main thing is that you can click on the buttons on the bottom to bring up various special test panels. These are more advanced but can come in handy for debugging. The Help Panel has really good detail in there on all of those things.

The MCU Test LED is a really handy thing to have, you can click on that and look to see if the Red LED goes on and off on the ZBC board. This is a good quick check to make sure that every thing is working OK between the PC and the PIC at least. The Reset FPGA let's you reset the FPGA from your PC remotely. It does not erase what is loaded or running on you ZBC. It is sort of the equivalent of doing the ole cntrl-alt-del on the old PC's. If you click on the FPGA Load pin, that will reset the FPGA and you will then need to reload the FPGA config.



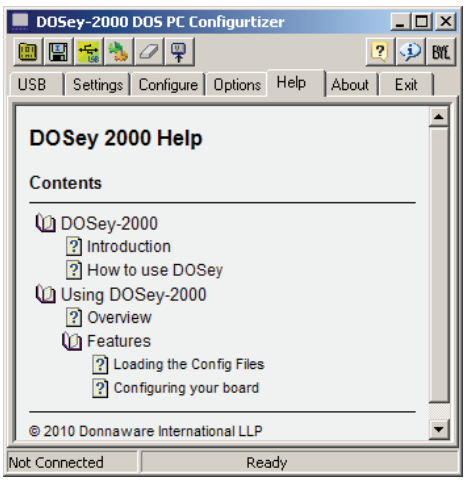
The **Configure** panel allows you to configure the FLASH and or FPGA on the ZBC. The top button "Upload RBF config File directly to FPGA" does just that, it uploads an RBF file on your PC over the USB up onto the FPGA without storing it on the FLASH RAM or anything like that. This is handy for testing new versions of Verilog HDL code if you are testing new versions. The next button below that will transfer an RBF file you previously loaded onto the FLASH RAM into the FPGA. Now, in order to get that RBF file into the Flash brings us to the next button, the one labeled "Upload RBF config file to FLASH RAM". Now, you can see all these buttons are labeled in a fashion that is pretty easy to figure out what they do, so there are no big mysteries here.

When you upload these files to the FLASH, you do not have to Erase the entire FLASH chip first or anything like that. It will automatically Erase the areas it needs to write back to. The Erase the entire chip is there just in case you want to wipe everything out and start over.



The next tab is the **Options** tab. On this one, you tell DOSey where to find the key files you need to configure ZBC. There are really just 3 main files you need to get it going. 1) the ZBC RBF FPGA configuration file, 2) the BIOS binary file and 3) the Floppy image file. Now technically you do not need the floppy image file. If you have a boot able SD CARD set up and ready to go, then it will boo off of that and you do not need the floppy image. I like having that because it is a nice fall back I think.

Once you set these paths up, then DOSey will always load those files straight from these locations. That way, you do not have to select the file every time. If you leave this step out, and just leave these blank or point to a non-existent file, then it will just prompt you for the file when you click on the button to upload something on the **Configure** tab.



The **Help** tab is real simple and straight forward, click on that tab and you get the regular help type structure as shown. From there you can click around and look things up.

When you click on the **Exit** tab, it waits about 5 seconds and if you don't click back to one of the other tabs then it just exits the program. You can also just click on the X in the upper right hand corner just like and Windows program and it will shut down. There is nothing special about having to click on Exit.

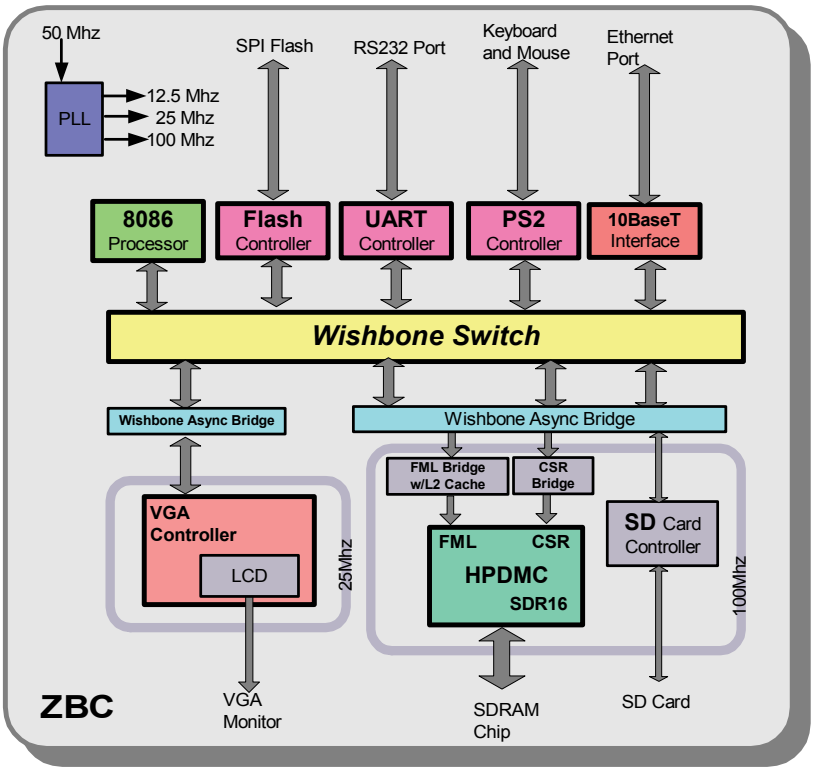
As for the USB interface, it is based on the HID layers so there are no drivers needed and no special set up or anything like that. You just plug SBZ in and assuming everything is assembled correctly, it should just work straight away. There is no need to open or close anything. In fact you can have DOSey open and turn on and off ZBC and it is no big deal, you will just hear the regular USB bong sound when it turns on and off.

Appendix F: ZBC Internal Architecture:

Before getting too far, it is really important to get an overall idea how the ZBC computer is put together. The architecture diagram below shows the components and how they are tied together using the Wishbone bus. The Wishbone bus is an Opencores initiative, you can download the specification at their web site (www.opencores.org). Also, please check out the web site (http://zet.aluzina.org) which is hosted by the inventor of the Zet processor, Zeus Gómez Marmolejo. There is a link to the forums for the Zet. I should mention, that simply purchasing a Terasic (www.terasic.com) DE1 or DE0 (cost is about $100 to $200) board and running Zet from aluzina.org is a good alternative for those who might be a little less adventurous from the hardware development perspective.

Care must be taken when crossing clock boundaries, that is what the Async Bridges are for, they allow you to connect a module running at one clock speed to a module running at a different clock speed. So we want to run our RAM and SD Card as fast as possible, so we run that at 100Mhz. For our VGA, that runs with a 25Mhz clock domain because then the timing of the horizontal and vertical sync pulses all works out. The main Wishbone bus and 8086 processor runs at 12.5 Mhz. If you are careful, you can run that part faster, I have successfully run it up to 24Mhz without any problems but to be conservative, I have it at 12.5Mhz which is a good place to start and then progress as you get things working.

The real heart of the system is the Wishbone switch in the middle, that is the clue that ties all the pieces together. If you take a look at the top level module ZBC.v, you see this right away. the wb_switch.v module has a number of ports set up. Each port has equal priority, The 8086 processor gets the Master port and all the other devices are on slave ports. If you look at the parameters that are passed to the wb_switch.v module, you see there is an address and address mask. This is how you tell it to map the peripheral or memory. Appendix D summarizes the assignments I am using.



If you wanted to add or remove a sub-system this is how you could do it. Let's say you want to remove the 10BaseT interface and replace it with a Game Paddle controller; you would remove the 10BaseT.v module from the Quartus zbc.qsf project and add in your game paddle module. Then remove the module instantiations from the top level module (zbc.v) and add in your own. Once that is done, then you can re-use the wb_switch port for your module. It is also possible to modify the wb_switch.v to add more ports if you wish.

The best way to understand how ZBC works is to dig into the code and start looking around. The following is a list of the modules that make up the system starting at the top:

File	**Module Description**
zbc.v	ZBC Top Level Module
cpu.v	8086 CPU
wb_abrg.v	Wishbone Async Bridge
wb_switch.v	Wishbone Async Bridge with registered outputs
pll.v	
BIOSROM.v	BIOS ROM
WB_SPI_Flash.v	Wishbone SPI FLASH RAM IO
fmlbrg.v	Fast Memory Link Bridge
hpdmc.v	High Performance Dynamic Memory Controller
csrbrg.v	SDRAM Control Bridge
sdspi.v	SD CARD SPI Interface
vdu.v	Text mode only VGA
simple_pic.v	Simple Priority Interrupt Controller
timer.v	Simple Timer
gpio.v	General Purpose IO
Sound.v	Sound Module
WB_PS2.v	PS2 Keyboard and Mouse Controller
WB_Serial.v	Wishbone RS232 Serial Controller
WB_Ethernet.v	Wishbone 10BaseT Ethernet Controller
seq_rom.dat	8086 Sequencer ROM
micro_rom.dat	8086 Micro code ROM
char_rom.dat	DOS Character Set ROM
biosrom.hex	BIOS ROM
xt_codes.dat	PC Keyboard Scan Code ROM Look up table

Appendix G: Game Paddle Experiment :

A good first example of a custom peripheral is to build our own game paddle. This is a brief tutorial on how to construct and install your own Game Paddle controller for ZBC.

How PC joystick port hardware works: The joystick port is a very simple 8 bit I/0 card which resides at I/O address 201h. The CPU can read and write to the joystick port I/O address 201h. Writing to that address starts joystick position measurement. Joystick interface only uses the write to the I/O address to reset the multi-vibrators in the card. The data value is not stored anywhere, so it does not matter what you write to the address

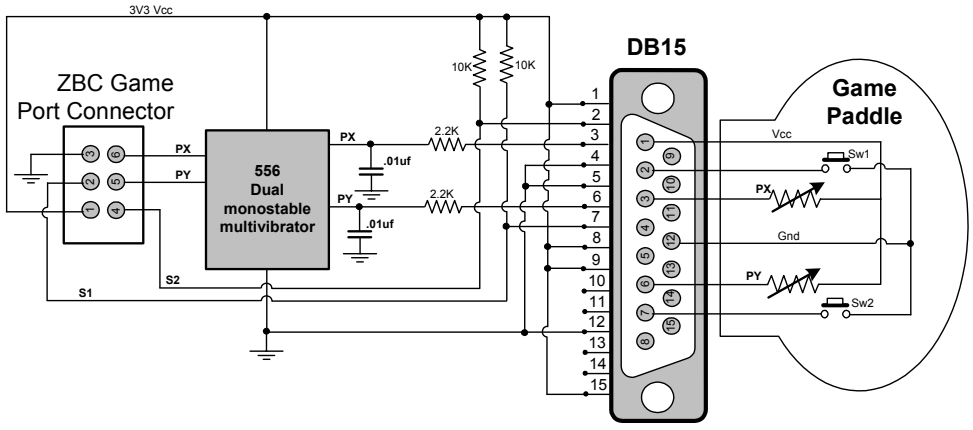
When you read one byte from I/O address 201h, you get the status information of the joystick interface. The following table will show how the bits are mapped in the value you get.

```
Game port 201h byte:
 _____ _____ _____ _____ _____ _____ _____ _____
|  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
| but4 | but3 | but2 | but1 | stk4 | stk3 | stk2 | stk1 |
|_____|_____|_____|_____|_____|_____|_____|_____|
```

The four most significant bits tell you the state of the joystick buttons. Four least significant bits tell the state of the multi-vibrators which are used for measuring the resistance value of the joystick position potentiometers. More accurate description of the bit meanings can be found at the table below:

```
7 6 5 4 3 2 1 0
* . . . . . . .  Button B2 (pin 14), 0=closed, 1=open (default)
. * . . . . . .  Button B1 (pin 10), 0=closed, 1=open (default)
. . * . . . . .  Button A2 (pin  7), 0=closed, 1=open (default)
. . . * . . . .  Button A1 (pin  2), 0=closed, 1=open (default)
. . . . * . . .  Mono-stable BY (from pin 13), 1=timing, 0=timed-out
. . . . . * . .  Mono-stable BX (from pin 11), 1=timing, 0=timed-out
. . . . . . * .  Mono-stable AY (from pin  6), 1=timing, 0=timed-out
. . . . . . . *  Mono-stable AX (from pin  3), 1=timing, 0=timed-out
```

For our implementation we are only going to implement joystick A, so the B bits will read zero.



Appendix H: Customizing the BIOS:

The BIOS is written using the Openwatcom compiler and assembler (available for free at www.openwatcom.org see, I told you everything is totally open on ZBC!) . Messing around with the BIOS is really one of the most advanced things you can do. Yes doing Verilog HDL is very specialized and seems scary at first, but once you get going on that, it is not too bad. But when you mess with the BIOS, you really need to understand what is going on with multiple levels; the hardware, the Verilog code, the overall architecture but also how DOS works and what it is doing and then on top of all that you need to know 8086 Assembler and also C code. Whew. So this is a super advanced topic. Probably the most important thing is to know DOS.

BIOS Stands for Basic Input Output System and is the abstraction layer between the hardware and the DOS operating system.

Appendix J: Copying and Attribution:

The ZBC is really an assimilation or compilation of a number of different components from many different sources. Yes, I did develop some things on my own, but the majority of the design is borrowed from other peoples projects. Every effort was made to insure that only open source or freely available sources are used. If anyone finds something that I have failed to provide attribution or might be infringing on intellectual property, please let me know. The intent of this project was to be completely open for educational purposes.

The following is a complete list of sources and attribution:

1. http://zet.aluzina.org Zeus Gómez Marmolejo, the Zet Processor
2. http://www.milkymist.org/ Milky Mist SoC
3. www.openwatcom.org OpenWatcom Compiler and assembler
4. www.opencores.org Open FPGA IP Core web site
5. www.terasic.com Terasic development boards
6. www.freedos.org FreeDOS project
7. www.fpga4fun Nice FPGA development boards and fun project
8.