



**ZFx86™**

**System-on-a-Chip**

**Training Book**

**Version 0.80 Rev A**

**December 4, 2001**

THIS DOCUMENT AND THE INFORMATION CONTAINED THEREIN IS PROVIDED "AS-IS" AND WITHOUT A WARRANTY OF ANY KIND. YOU, THE USER, ACCEPT FULL RESPONSIBILITY FOR PROPER USE OF THE MATERIAL. ZF MICRO DEVICES, INC. MAKES NO REPRESENTATIONS OR WARRANTIES THAT THIS DATA BOOK OR THE INFORMATION CONTAINED THERE-IN IS ERROR FREE OR THAT THE USE THEREOF WILL NOT INFRINGE ANY PATENTS, COPYRIGHT OR TRADEMARKS OF THIRD PARTIES. ZF Micro DEVICES, INC. EXPLICITLY ASSUMES NO LIABILITY FOR ANY DAMAGES WHATSOEVER RELATING TO ITS USE.

### **LIFE SUPPORT POLICY**

**ZF MICRO DEVICES'** PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZF MICRO DEVICES, INC.

As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

(c)2001 ZF Micro Devices, Inc. All rights reserved.

ZFx86, FailSafe FailSafe Boot ROM, Z-tag ZF-Logic, InternetSafe, OEMmodule SCC, ZF SystemCard, ZF FlashDisk-SC, netDisplay, ZF 104Card, ZF SlotCard, and ZF Micro Devices logo are trademarks of ZF Micro Devices, Inc. Other brands and product names are trademarks of their respective owners.

Note: Fonts in this book are large in that the book is designed to be used in classroom lecture with an overhead projector.

<b>Chapter 1 - Overview .....</b>	<b>7</b>
Finally a real X86 System on a Chip .....	8
Introducing the ZFx86 .....	9
ZFx86 - The Integrating Platform .....	10
ZFx86 Features .....	11
The Chip and the Data Book .....	12
The Block Diagram .....	13
Block Diagram: ZFx86 Vs. RISC .....	14
Pentium vs 486 Core CPU Technology .....	15
The ZFx86 Integrated Development System .....	16
ZFx86 “toy” Board Demonstration Design .....	19
Tri-M Systems MZ104 PC/104 ZFx86 Board .....	21
ZFx86 Documentation .....	26
e-Commerce Server / Companion Chip .....	27
<b>Chapter 2 - x86 Processor .....</b>	<b>28</b>
ZFx86 Block Diagram .....	29
Selling The ZFx86 x86 32-Bit CPU .....	30
Chip Benefits Overview .....	31
Power Management .....	32
Memory Address Space – SDRAM .....	33
Memory Address Space – ISA, PCI, ZF-logic .....	34
Interrupts and the RTOS .....	35
Task State Transitions in VxWorks RTOS .....	36
8259A PIC Interrupts .....	37
NMI, SMI and SCI .....	38
Interrupt Vector Assignment .....	39
Real Mode (DOS) .....	40
Extended and Expanded Memory in Real Mode .....	41
Basic Protected Mode Operation .....	42
Privilege Levels – Data Access .....	43
Privilege Levels – Code Access .....	44
MMU – Page Directory and Page Tables .....	46
MMU – Translation Look-Aside Buffer .....	47
IOPL and SMM Protection .....	48
On Chip – L1 Cache .....	49
Integrated FPU .....	50
<b>Chapter 3 - North Bridge .....</b>	<b>53</b>
ZFx86 Block Diagram .....	54
North Bridge Features .....	55
North Bridge Features Q&A .....	56
North Bridge Overview .....	58
North Bridge SDRAM Controller .....	59
North Bridge PCI .....	61

North Bridge Cache Management .....	62
Bus Arbitration on the PCI Bus .....	63
PCI vs. ISA bus mastering .....	64
The Role of the NB in Power Management .....	65
Answers to Questions .....	66
<b>Chapter 4 - South Bridge .....</b>	<b>68</b>
ZFx86 Block Diagram .....	69
Definition of Functional Blocks .....	70
Bus Comparison .....	71
Front Side Vs Back Side PCI .....	72
Bus Mastering IDE Controller .....	73
USB.ORG Website .....	74
Super I/O .....	75
Serial Port .....	76
Infrared Communication Port .....	77
Parallel Port .....	78
System Wake Up Control .....	79
Access (I2C) Bus .....	80
Real Time Clock .....	81
Real Time Clock Alarms .....	82
<b>Chapter 5 - ZF-Logic .....</b>	<b>83</b>
ZFx86 Block Diagram .....	84
ZF-Logic Block Diagram .....	85
ZF-Logic Register Space Access (8-Bit) .....	86
ZF-Logic Register Space Access (16, 32-Bit) .....	87
ZF-Logic Registers .....	88
ISA Memory Windows for Flash / SRAM .....	91
Benefits of Memory Window Mapping .....	92
Safety Aspects of Memory Windows .....	93
Memory Window Registers .....	94
ZF-Logic Memory Windows Review Questions .....	95
GPCS I/O Mapper .....	96
GPCS (I/O Mapper) Register Set .....	97
GPCS (I/O Mapper) Control Registers .....	98
Watchdog Timer .....	99
ZF-Logic Registers for the Watchdog Timer .....	100
External Control of Watchdog Timeout .....	103
PWM Generator .....	104
PWM Generator - I/O Control Register .....	106
Boot Parameters Register .....	107
Boot Parameters and Clocking .....	111
Answers To Questions .....	117



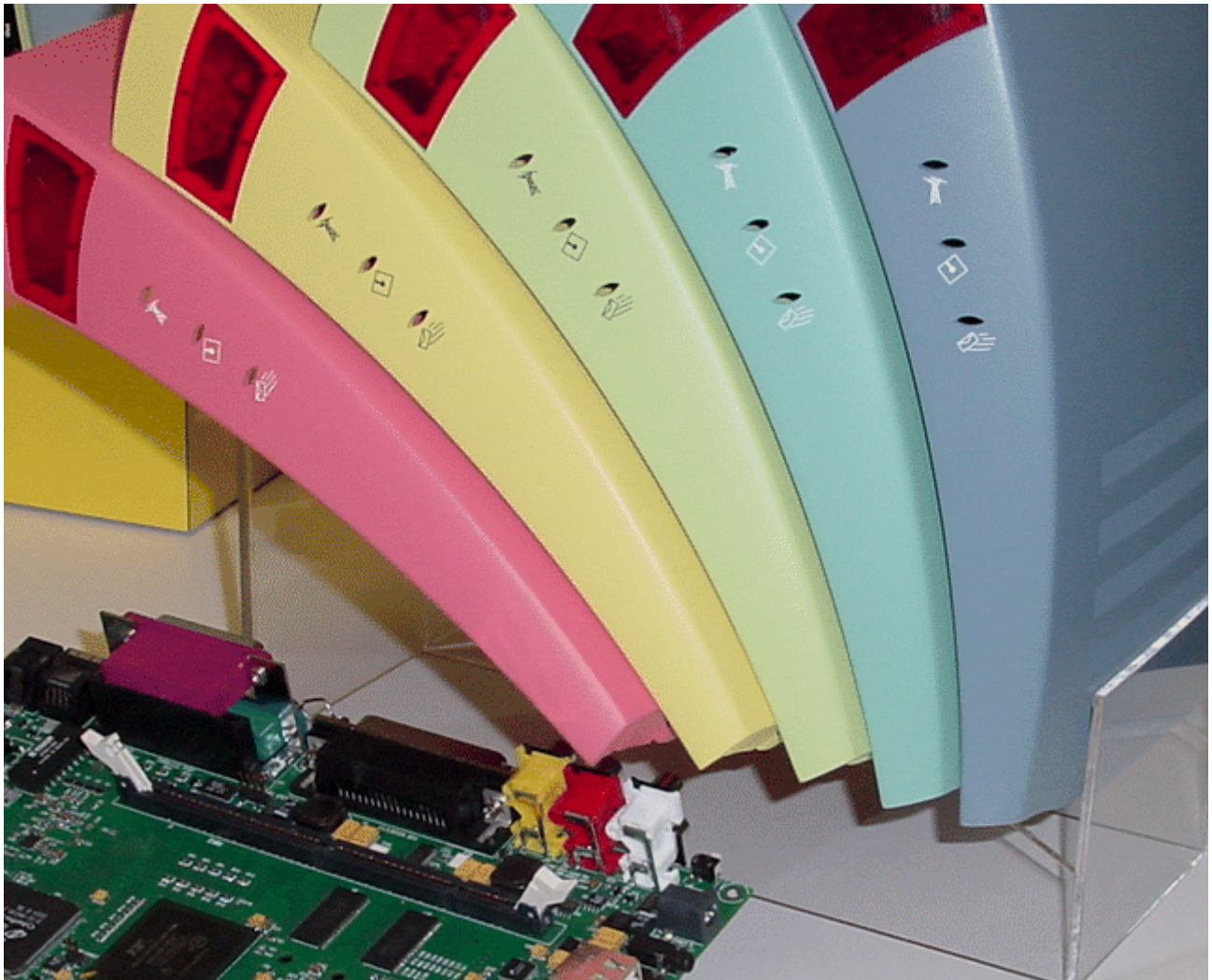
**Chapter 6 - Z-tag and BUR ..... 121**

- ZFx86 Block Diagram ..... 122
- Using The Z-Tag Manager ..... 123
- The Z-tag Dongle ..... 124
  - The Ztag Dongle (Continued) ..... 125
  - Normal vs. PassThrough Download Mode ..... 126
- The Z-tag Manager Interface ..... 127
  - Z-tag Manager Commands ..... 128
  - Z-tag “Memory” Dongle Programming ..... 129
  - Z-Tag “PassThrough” Dongle Programming ..... 130
  - Put the BIOS in Dongle - First Get Flash Program ..... 131
  - Editing Command 01 - Upload & Execute Code ..... 132
  - Edit Selection of Flash Code, Add BIOS Basket ..... 133
  - Add The Stop Command, Create A “Save” Folder ..... 134
  - Copy and Paste Commands to Work Area ..... 135
  - Copy Program Through the PassThrough Dongle ..... 136
  - Test by Reading Back from the “Memory” Dongle ..... 137
- BUR Version Test Program Source Code ..... 138
  - BUR (Fail-safe) Boot Up ROM ..... 139
  - Basic Component Initialization ..... 140
  - Elementary debugger console functionality ..... 141
  - Data Fetch and Execute ..... 142
  - Basic OS functionality for user code ..... 143
- Manufacturing and Field Tools ..... 144
  - BUR: Debug tool ..... 145
  - Fail-safe System ..... 146



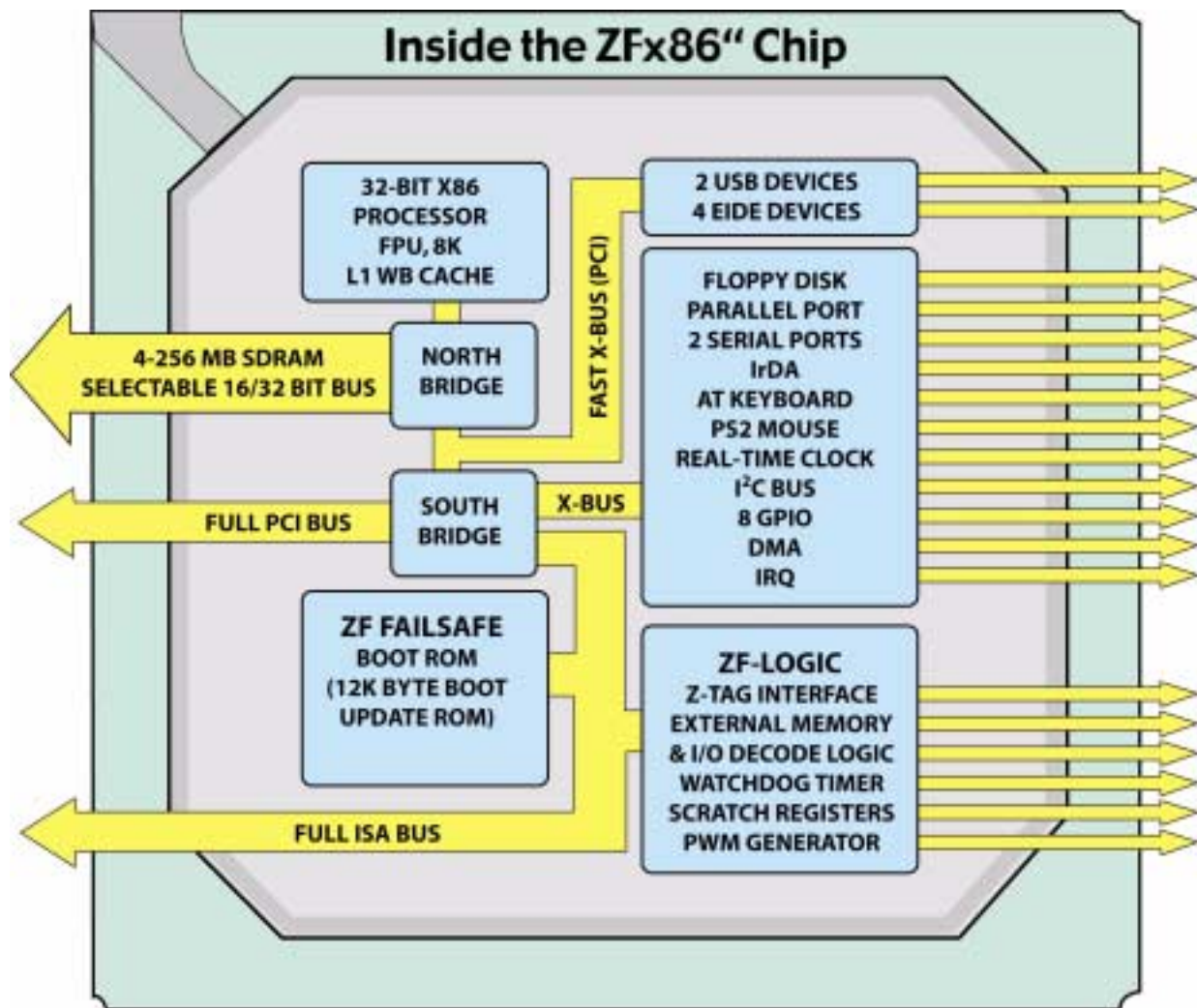
# ZFx86 Training Book

## Chapter 1 - Overview



NOTE References compatible with ZFx86 Developers Data Book version 0.80.

## 8 Finally a real X86 System on a Chip



- **Processor:** 486+ CPU at 128 MHz
- **North Bridge:** DRAM Controller and FrontSide 64 MHz PCI Bus
- **South Bridge:** Generates BackSide PCI and ISA Buses.
- **USB + Extended IDE Device Interface:** on the FrontSide PCI Bus
- **SuperIO:** Industry Standard X86 I/O + I<sup>2</sup>C
- **ZF-Logic:** ZF Additions for Embedded Systems, Low BOM cost, and FailSafe

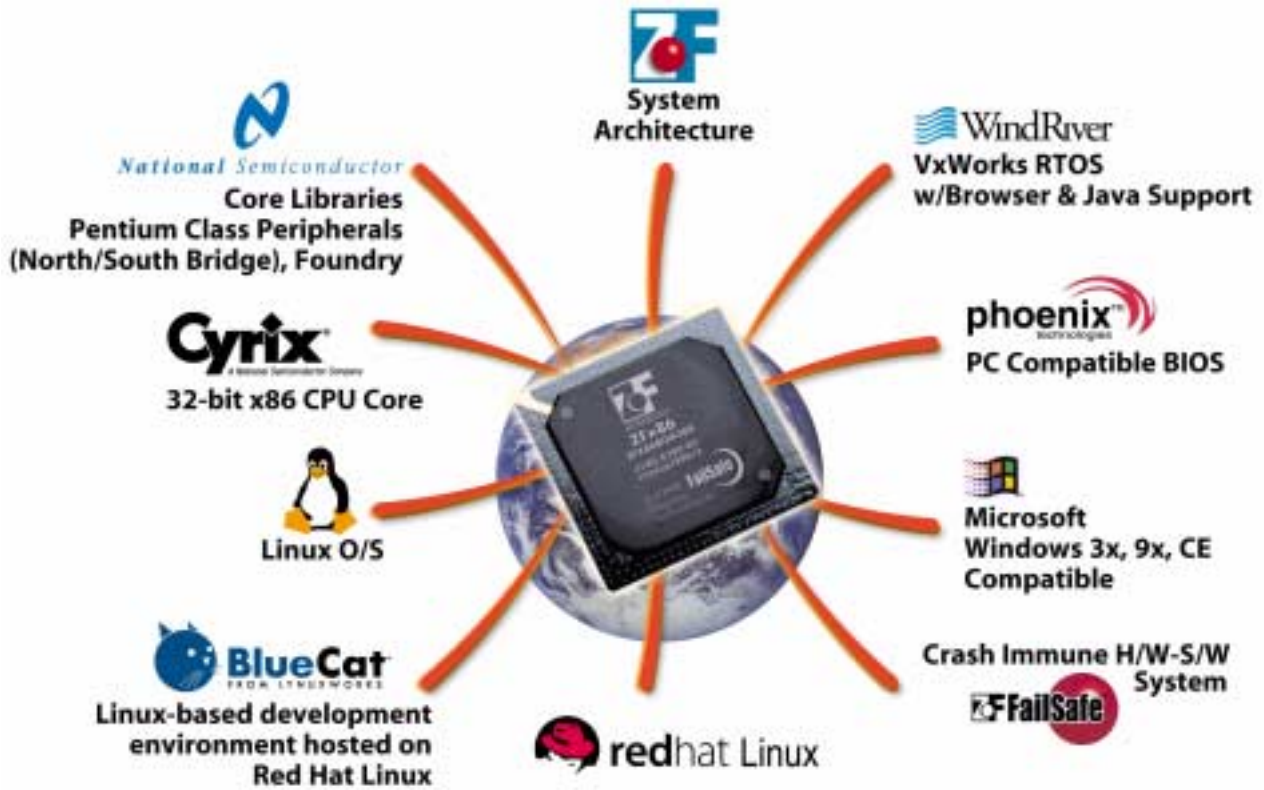


### Introducing the “FailSafe ZFx86” for the next generation of embedded systems

- Unequaled set of traditional PC H/W features
- Lowest BOM cost in market for lowest OEM product cost
- System level architecture to minimize integration complexity
- ZF H/W features unique to embedded market
- Bundled S/W & Firmware completes PC "system"
- Mainstream .24 micron technology
- “Proven” industry standard architecture
- World class silicon partner



# 10 ZFx86 - The Integrating Platform



## 11 ZFx86 Features

---

1. .5 Watts @ 100MHz Typical
2. License free BIOS supporting Windows 9X, CE, DOS, NT/E, Linux
3. Cyrix 586 FP DX 32 bit core with Pentium class North/South Bridge
4. Dedicated PCI & ISA Bus - no multiplexing, supports legacy ISA code
5. USB fully supported
6. SDRAM fully supported
7. Selectable 16/32 bit DRAM Bus reduces memory bloat
8. Patented FailSafe™ system for crash immune operation
9. Fully PC compatible; runs all x86 code NATIVELY
10. Robust Integrated Development Platform S/W Installed - DOS, Linux, RTOS, Full BIOS Included

## 12 | The Chip and the Data Book

---

The first thing to remember is that if someone asks you “**is it inside the chip?**”, the answer is “**Yes**”.

There is so much inside this chip that you'll be hard pressed to find something that is not there.<sup>1</sup>

The second thing you have to remember is that **your best friend is the data book**. The data book is over 600 pages long, and covers in detail almost everything we were going to talk about. Most of the stuff in this training book is just right out of the data book. If you have a question, go to the data book, it's there.

---

1. A Pentium Processor and an Ethernet Controller and a Video Controller have been left out of the chip. The Pentium would have made the entire chip far too complex and expensive -- our enhanced 486/133 with ISA and ISA and Super I/O (etc) provides lots of functionality and performance.

An Ethernet Controller and Video Controller would have again driven the price and complexity far too high. However, ZF plans a companion chip to add the Ethernet and Video. See ['e-Commerce Server / Companion Chip' on page 27](#).



## 13 The Block Diagram

---

The first reference point is the block diagram. That's what the chip is all about.

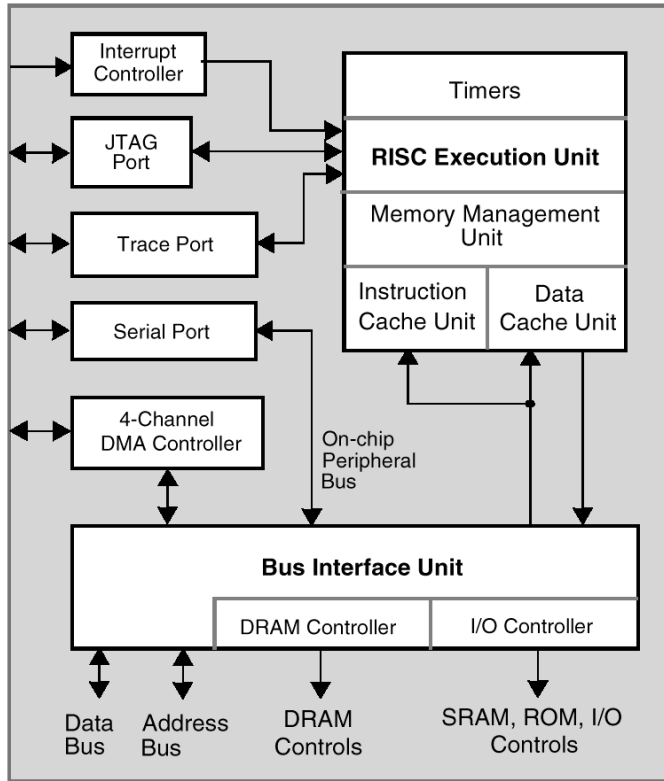
The first thing we're going to talk about is the **processor core**. We will step through what is in there. The processor core talks to what we call the North Bridge.

Then we will talk about what is in the **North Bridge**. What things the North Bridge does, how it tries to keep track of who's talking to whom. The North Bridge talks to the rest of the system through a **Front Side PCI bus**. This is an internal bus; none of these pins come out to the outside world. I think it interesting and important to note that this is the way it is structured inside, because some of the devices are actually right on the internal bus. To the outside world, you have a **Back Side PCI bus**.

Talking to the **South Bridge** through the ISA bus is the **Super I/O**. These are traditional PC things, things that we are used to seeing everywhere.

Everything below the **Super I/O** is **functionality that we have added to the chip** that is beyond the normal PC architecture. We call this the **ZF Logic, the BUR, and the Z-tag**. These are things that we find interesting to the embedded designers, things that people need to put together their systems with less cost and less pain than normally is associated with designing a CPU into an embedded device.

# 14 Block Diagram: ZFx86 Vs. RISC



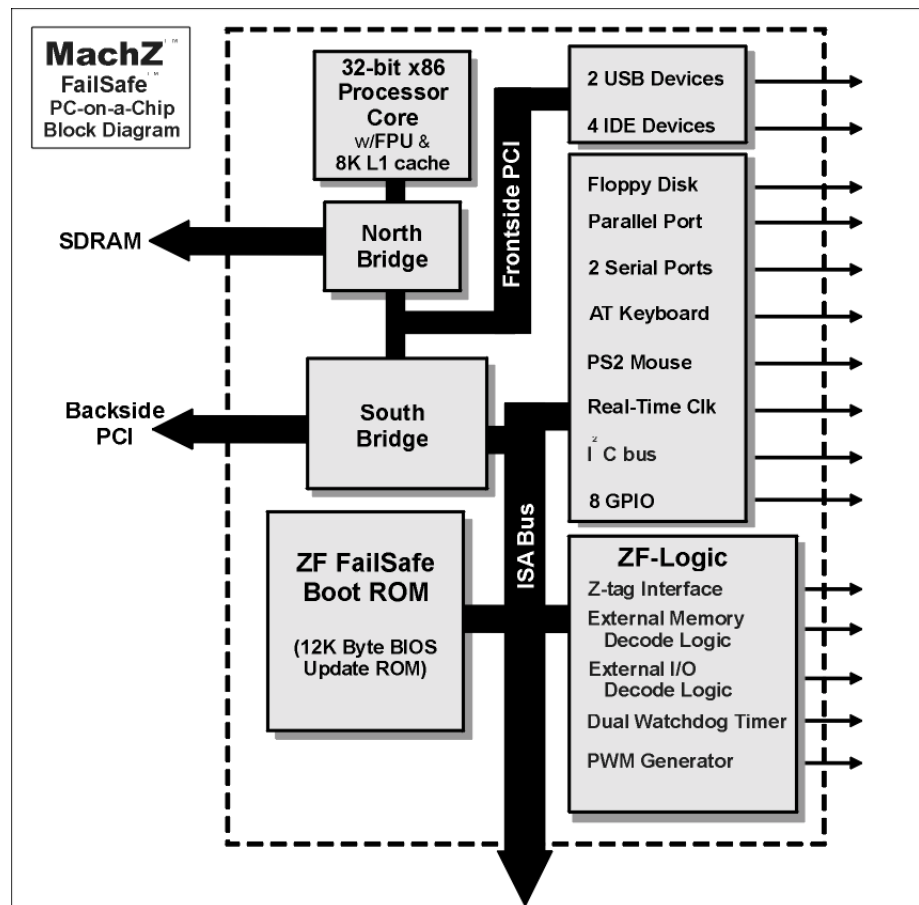
**RISC Peripheral Integration Density** is much lower than ZFx86

**Time-to-Market:** x86 is always first platform supported due to Wintel dominance

**Legacy x86 HW Base** will be difficult for RISC to overtake (Ethernet, Modems, Graphics Controllers, HomePNA, etc.)

Integrated RISC Designs require **custom or semi-custom chip development** and huge \$ investment in design tools.

Expensive and time consuming: Only available to major corporations RISC Firmware, **O/S and Application development is longer** & requires unique expertise



# 15 Pentium vs 486 Core CPU Technology

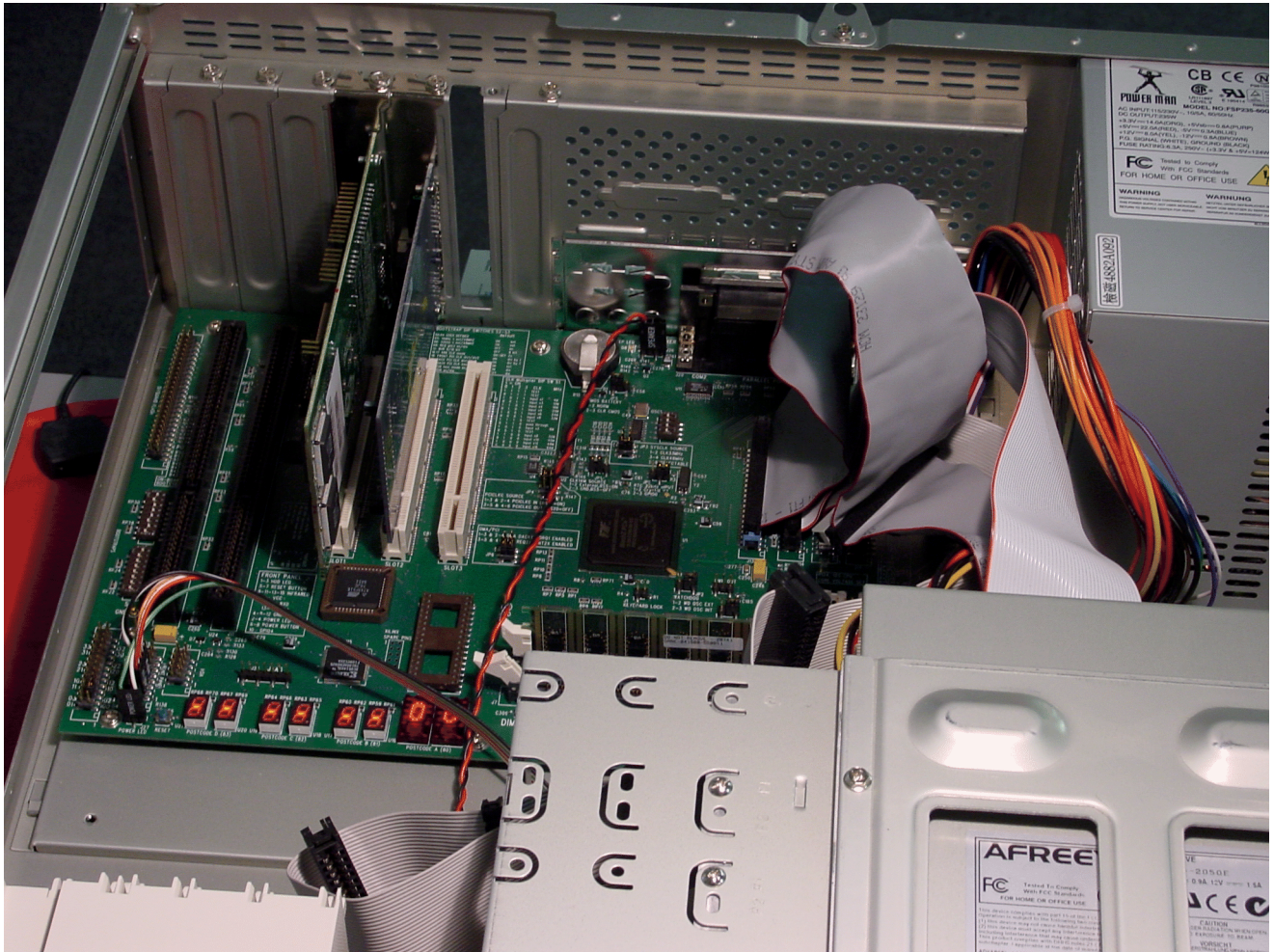
**Table 1: Increasing Power - More and More Devices/Chip**

CPU	Transistors	Layers	Width	Size
8080	3,000			
8085	6,500			
8086/8088	29,000			
80286	60,000			
80386	275,000	10	1 um	257
80486	1,200,000	12	.8 um	357
Pentium	3,100,000	18	.6 um	506

CPU	Transistors	Layers	Width
ZFx86 486+	2,500,000	20+	.24um
ZFx86 Pentium <sup>a</sup>	4,500,000		

a. There is no ZFx86 Pentium. The point of the diagram is to show that the “sweet spot” in performance Vs. transistors is the 486+.

# 16 | The ZFx86 Integrated Development System



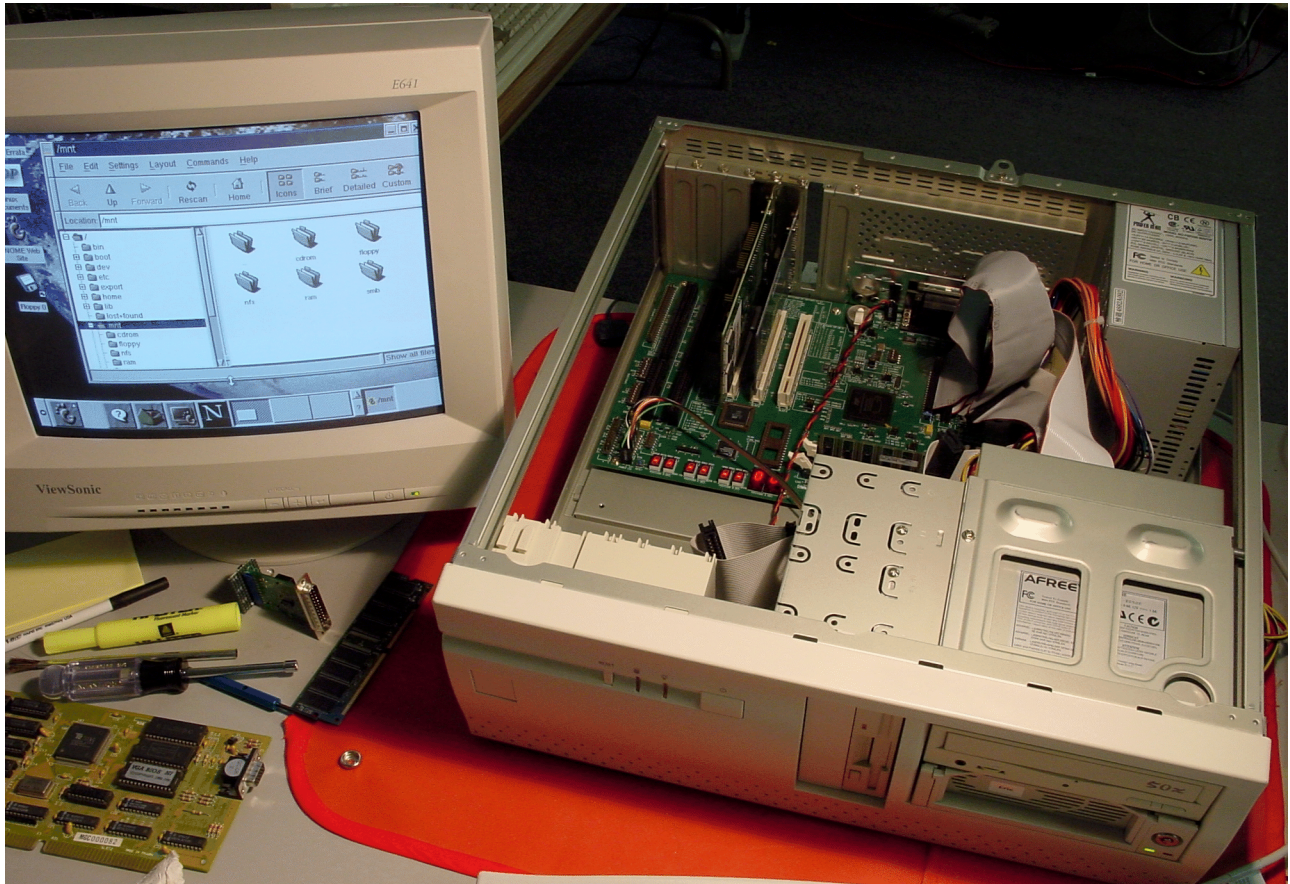
ZFx86 32bit CPU  
10/100Base-T Ethernet  
VGA/XGA/SVGA/SXGA  
CRT controller  
Two serial port connections, one  
selectable as IrDA  
One parallel port connection  
Three PCI expansion slots  
Two ISA expansion bus slots  
Two USB ports  
PS/2 keyboard and mouse  
Floppy disk drive  
IDE (ATA-4) disk drive

CD ROM disk drive  
OS software and utilities  
preloaded

Lynx Real-Time Systems, Inc.  
Blue Cat Linux  
Full Red Hat  
Linux 6.X distribution  
Phoenix BIOS  
Caldera DR-DOS  
Wind River VxWorks RTOS



# 17 The ZFx86 Integrated Development System



The ZF ZFx86 Integrated Development System is a **network-ready** and **video-ready**, full-function ATX size evaluation system. The system features ZF's system-on-a-chip, ZFx86, with **10/100Base-T Ethernet (PCI) card** and a **display controller (PCI)** to create an evaluation environment that allows the OEM designer to test the ZFx86 processor with proprietary hardware and software.

## Complete Feature Set

The ZF ZFx86 Integrated Development System incorporates all the functionality of a standard PC motherboard with a number of enhancements and added features. The board includes **serial and parallel connectors, floppy disk header and IDE connectors, user-available flash, external JEDEC byte-wide socket, ISA and PCI expansion bus connectors**, and a **PC-compatible BIOS**. Our patent pending **FailSafe™ Boot ROM** allows the user to easily reboot the system if the BIOS is corrupted, or if the CMOS is inadvertently configured in a manner which locks the system in an unusable mode. The user can then simply reprogram the FLASH to recover the system. In addition an **ATX power supply**, a **hard drive** (preloaded with an RTOS and Linux OS), a **floppy drive, a CD, a keyboard, mouse, and cables** are all included to facilitate the engineer's bring-up task.

## Quicker design cycle, shorter time-to-market

Easily integrate the widest selection of embedded hardware peripherals by attaching ISA or PCI expansion cards directly to the board via the sockets provided. The PC/AT ROM-BIOS and OS enable you to develop software on your desktop PC and then easily transfer your development work to the embedded system with little or no modification. Or you can develop your application software directly on the ZF ZFx86 Integrated Development System.

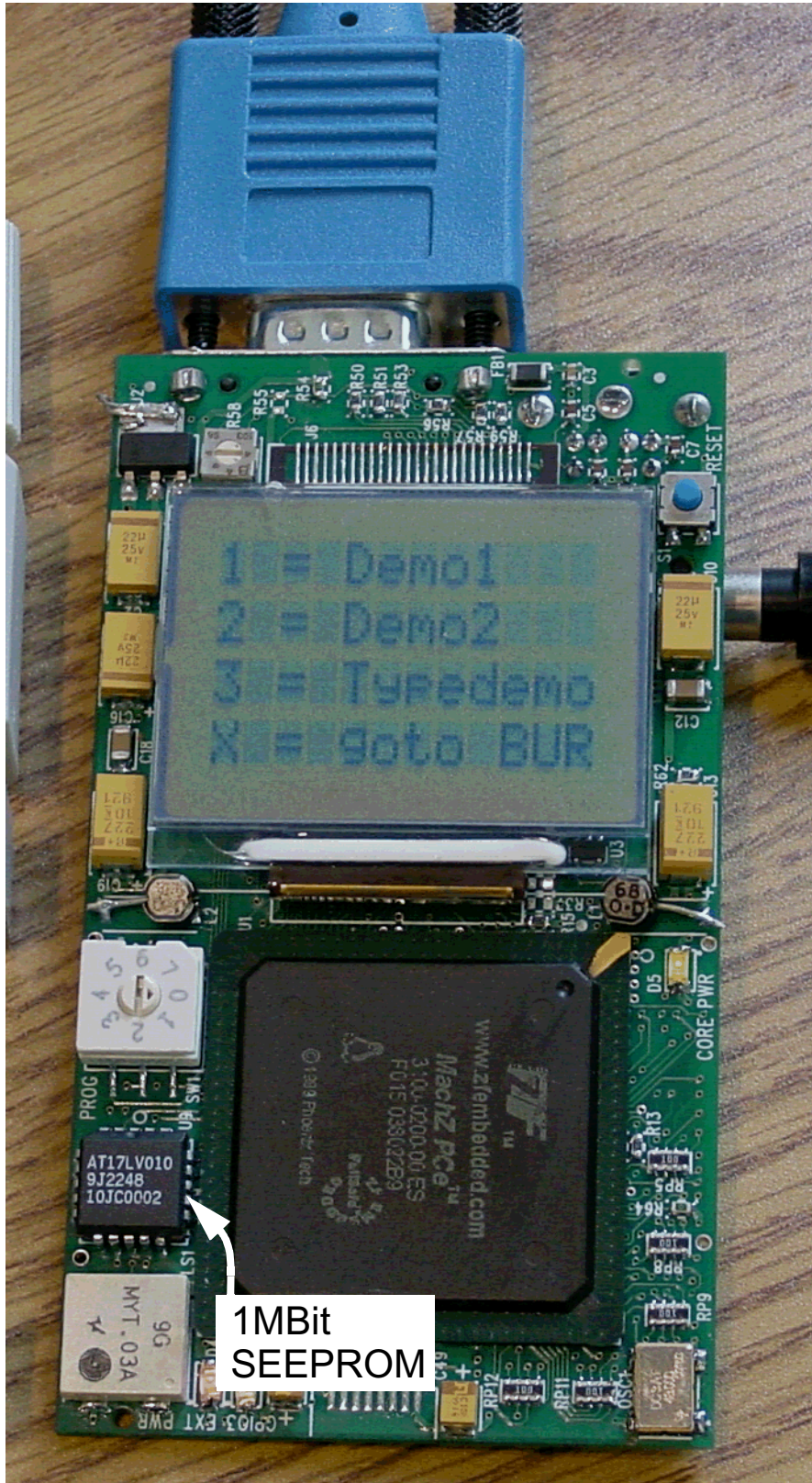


# 18 The ZFx86 Integrated Development System





# 19 ZFx86 "toy" Board Demonstration Design



0 DRAM 0 FLASH  
BUR Demo

Demo Programs are transferred to an on-board SEEPROM using the Z-tag manager software.

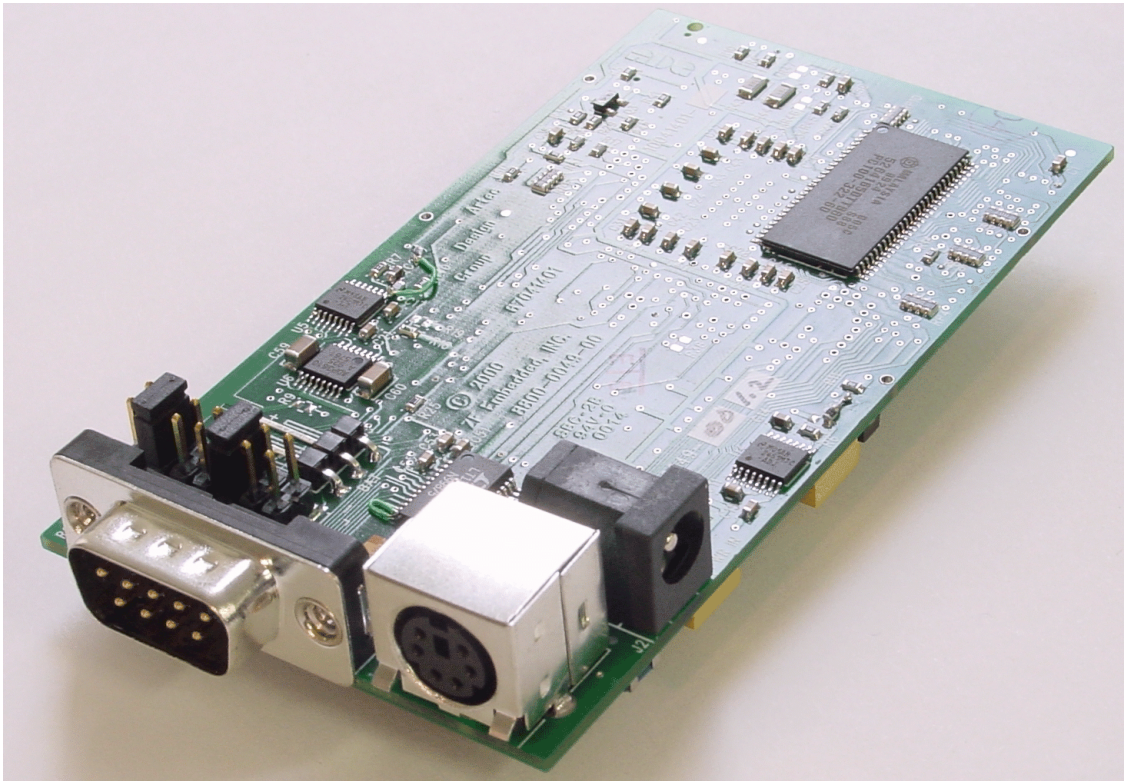
On power up, the program is read into the ZFx86 on Chip RAM by the BUR.

8MB DRAM  
16 MB Flash  
pocket PC

1Mbit on board SEEPROM can be used as an on board Dongle.

On power up, the program is executed out of 16 MB FLASH with 8MB External DRAM.

## 20 Toy Board Continued

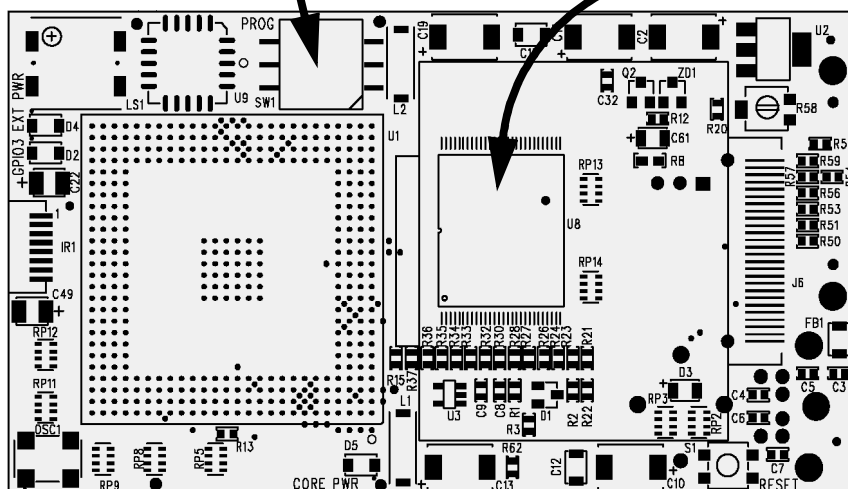


### Toy Board with ZF<sub>x</sub>86 System on a Chip

- on-chip keyboard/mouse controller
- on-chip SDRAM Controller
- on-chip Flash Chip Select (Memory Mapper)

1 Mbit SEEPROM

16 MB Flash





## 21 | Tri-M Systems MZ104 PC/104 ZFx86 Board

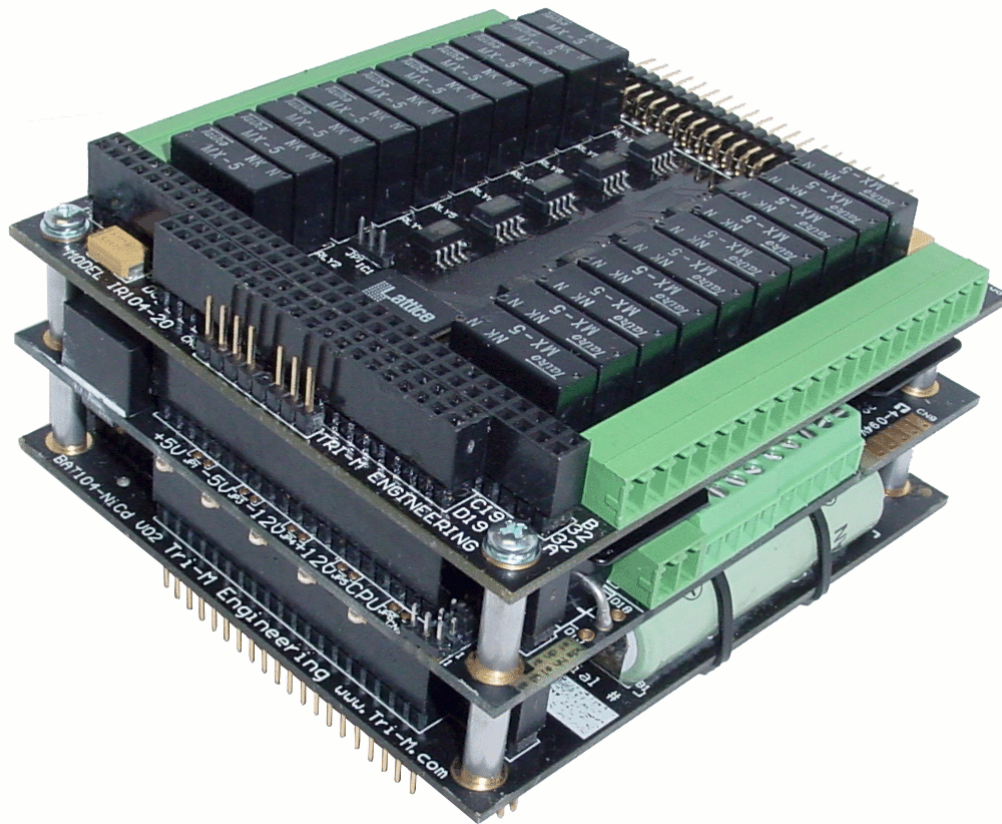


The MZ104 board is fully compliant to the PC/104 specification. Using the ZFx86 on a 6 layer circuit board. The MZ104 brings out most of this chips' internal features, and supports a 2 or 4 Megabyte Flash ROM, the M-Systems DiskOnChip, and a 144 pin SO\_DIMM connector supporting either 32 or 64Mb of fast SDRAM. The MZ104 is the first and least complex member of a family of three PC/104 boards designed around the ZFx86.



See [www.controlled.com/pc\\_104/consp5.html](http://www.controlled.com/pc_104/consp5.html)). This means that it is part of a family of about 5000 boards produced by about 160 suppliers (see [pc104.org](http://pc104.org)).

## 22 PC/104 Module Stack



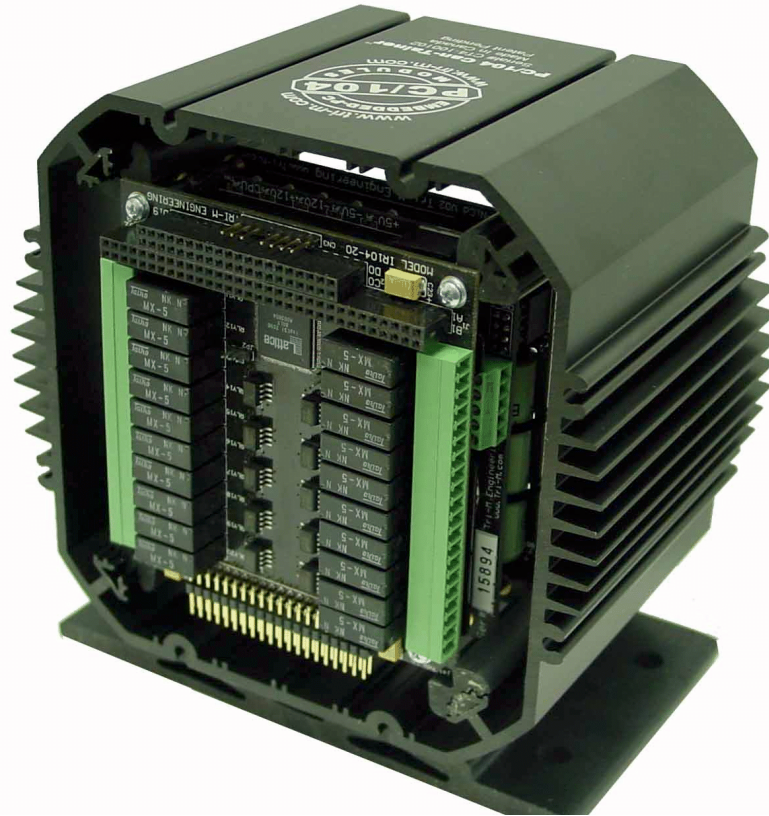
PC104 modules will stack together connecting the bus from one card to the next. In this case, the top card is a relay board showing the somewhat industrial history of the PC/104 modules.

In the middle of the stack we have a CPU card, and on the bottom there is a UPS battery board.

The PC/104 connectors themselves use long pins and deep sockets which provide a very reliable electrical interconnect and which act as a spine to strengthen the stack. Nylon or metal spacers are used on the corners adding strength and rigidity to the stack. Compare this to standard ISA cards, which are prone to intermittent electrical connections -- especially when in motion!



## 23 PC/104 Enclosure

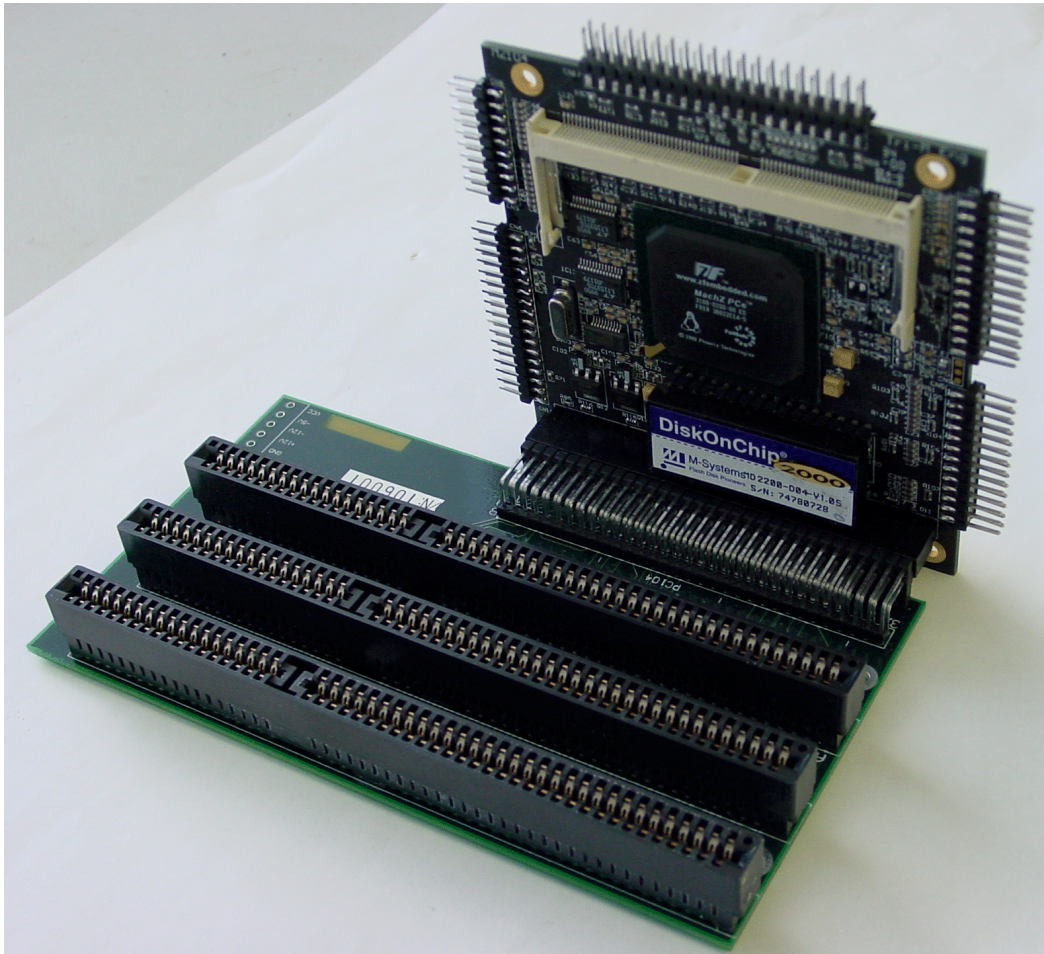


Pictured here is an aluminum enclosure with rubber shock absorbing mounts designed specifically for hostile environments.

Enclosures for PC/104 are basically there to protect the electronics from the environment. They are also designed to shed as much heat as possible to keep the internal electronics cool. Good designs also include consideration of vibration and G-force minimization to protect the PC/104 stack when mounted in high vibration or G-force locations.

Marketing people like the idea of selling a black box solution whose price is justified by its function rather than its component costs. These high stamina enclosures provide added protection and thus added value.

## 24 Three Slot 16-Bit ISA Bus Passive Backplane



This picture shows a MZ104 plugged into a three slot standard 16-bit ISA bus passive backplane. Backplanes of this type are available from various manufacturers.

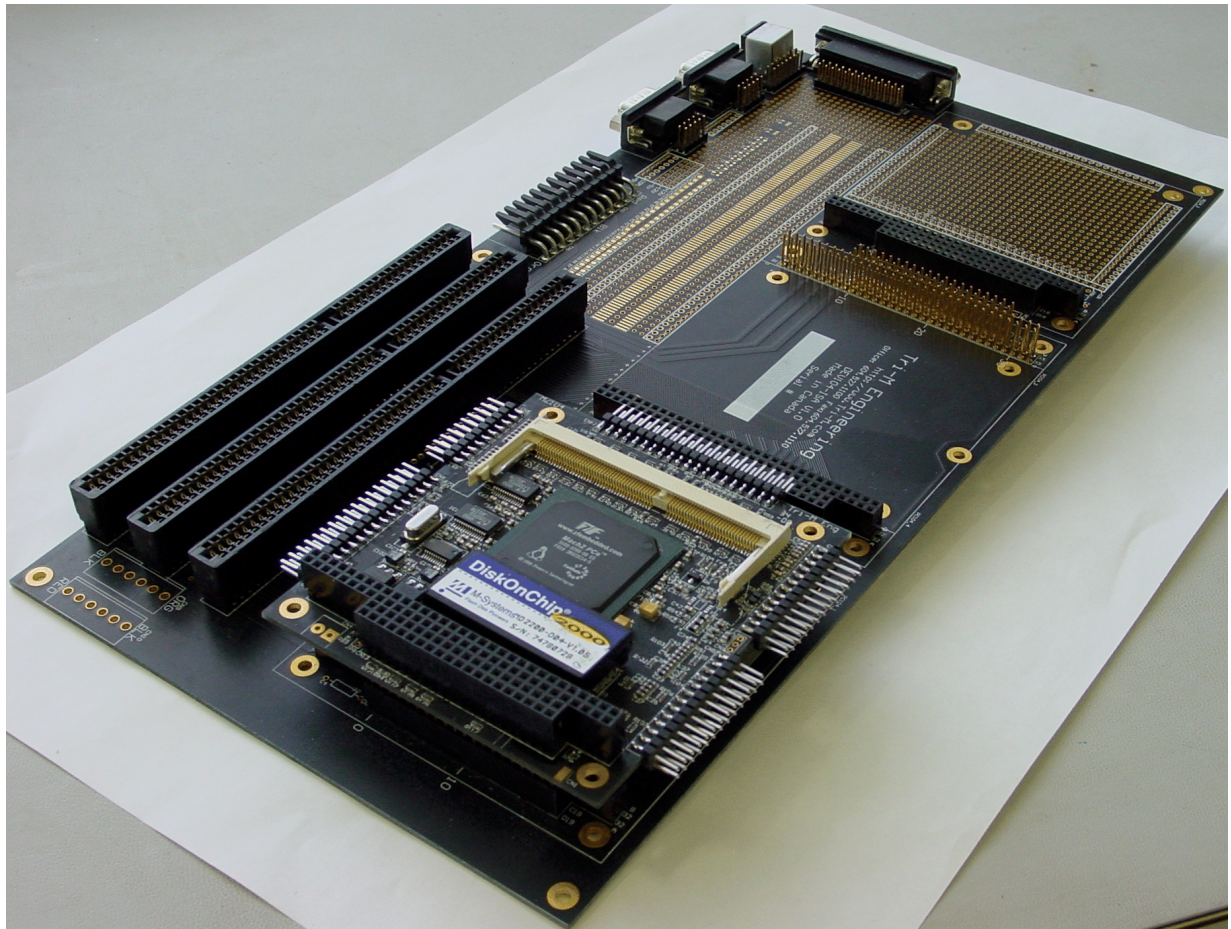
Backplanes such as this allow you combine ISA cards and PC/104 cards during prototyping and testing. Although only the MZ104 card is shown in the figure, a complete PC/104 stack could be connected to up to three ISA cards using this backplane.

The backplane allows for easy testing of compatibility of features and functions in chip sets under consideration for design into your embedded solution.

When would you use a passive backplane? If, for example, you had five different frame grabber chip sets, you could purchase ISA cards which use these chips and then test and implement your software without having to build custom boards of your own using each of the chips under consideration.



## 25 Development Backplane Board

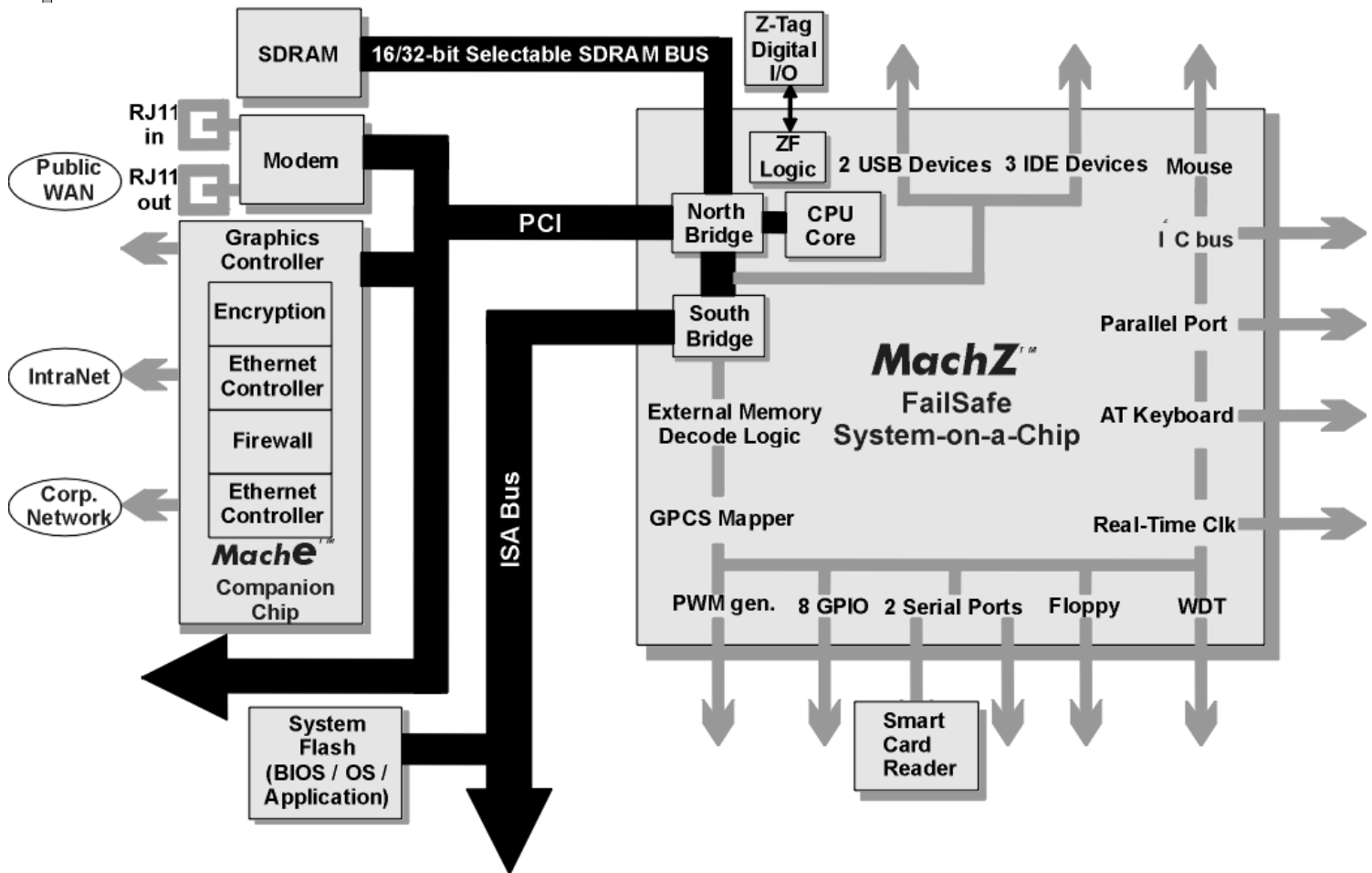


The Tri-M DEV 104 Development Backplane Board allows not only for the connection of PC/104 and ISA cards, but provides hardware prototyping areas for both thru-hole and surface mount electronic components. This is very useful for solution providers who require the development of custom boards based on a hardware prototype. In addition to the PC/104 header that the MZ104 is mounted on, there are three additional PC/104 connectors, one of which has male pins facing upwards which allows testing of components on the back side of a PC/104 card. This is much easier than testing in a stack!

<b>Filename</b>	<b>Description</b>
ZFx86 Data Book.pdf	Complete Data Book.
ZFx86 Training Book.pdf	Contains many details of the unique features of the ZFx86 device, including the Dongle. This book.
ZFx86 Integrated Development System Quick Start Guide.pdf	Top Level User Guide for the ZFx86 Integrated Development System
Annotated Evaluation 1 Board Schematic.PDF	Development System board schematic with comments to clarify different aspects of the board.

Note: These documents are provided on a CD which accompanies the ZFx86 Integrated Development System.

## 27 | e-Commerce Server / Companion Chip



ZF is investigating chip sets which would increase the value of the ZFx86.

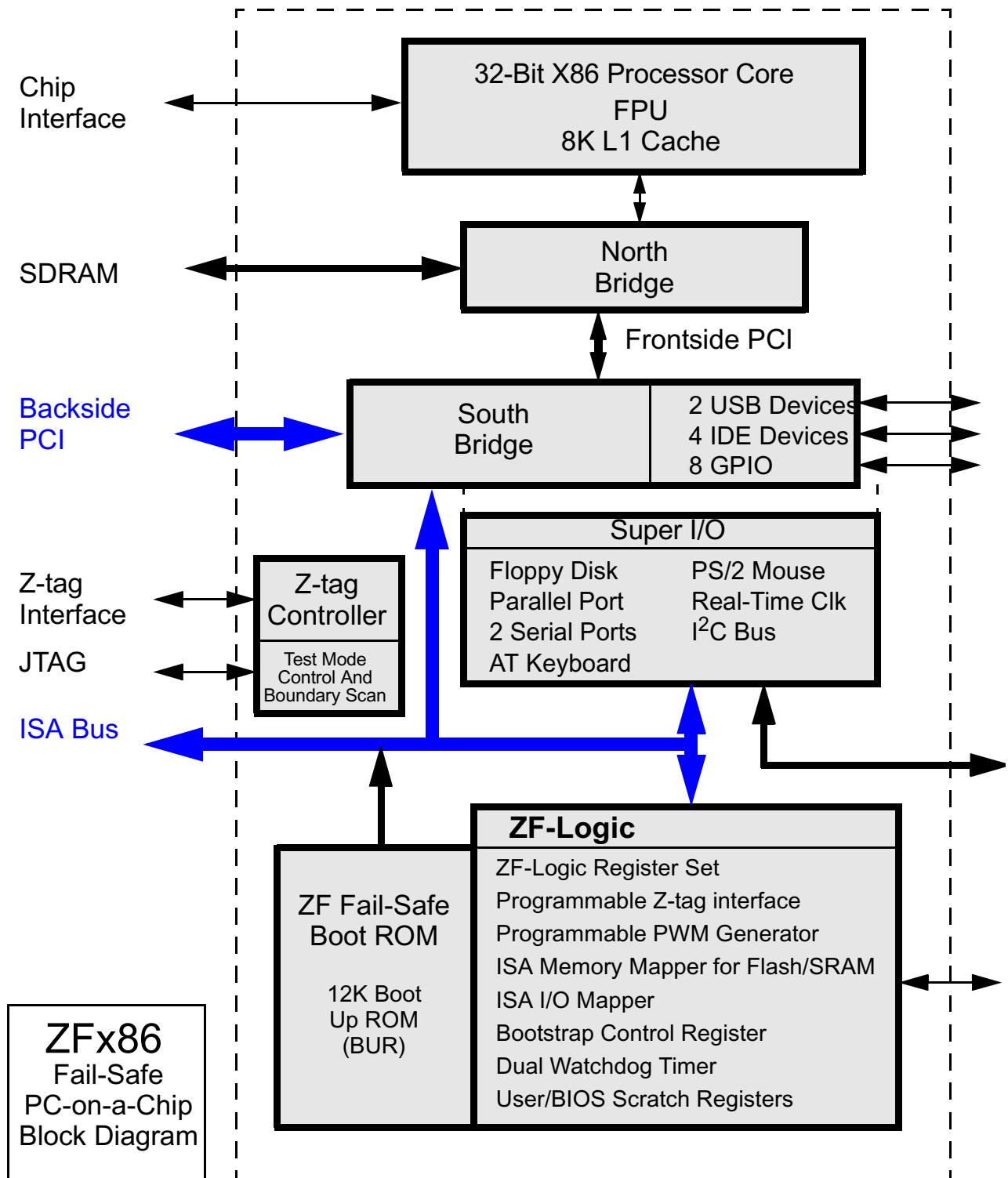
A possible companion chip with Graphics and Ethernet ++ might provide a 2-chip solution for 150% of the cost.

# ZFx86 Training Book

## Chapter 2 - x86 Processor



# 29 ZFx86 Block Diagram



The CPU: a lot of these things that you see here are X86 and PC buzzwords that you have seen before. It is in the manual and it is in the chip.

We will not always be compared to another x86 processor.

When you go out into the field you'll find there are **two groups of people**: those people who have decided to **x86 is theirs forever**, they love it and they want it, and they are going to ask you one set of questions.

Then there's going to be a set of people who say "**I have decided to go with a RISC processor**",. Or "what this Transmeta thing all about": what is the Caruso<sup>1</sup> chip all about? We are going to go into that later.

---

1. See ['Why Not Crusoe' on page 52](#).

## 31 | Chip Benefits Overview

---

A lot of the stuff that we point out here shows **this chip has got it**, and it's a particular attribute of the x86 architecture. Essentially we're picking up **20 years of legacy**. All the best minds in the valley: Intel, AMD, big companies, have been working on this architecture for the past 20 years. We just inherited from a "rich Uncle", and that's what we're bringing to you.

**Power management** is a hot button everywhere. We will talk about how power management works through the chip; **power management has to span the entire chip, from the top to the bottom.**

We challenge Transmeta to have all the **addresses spaces** that we have.<sup>1</sup>

Interrupts **can be tricky, especially when you put PCI onboard.**

**Write-thru vs. write back cache** is explained in the manual very clearly. Everyone knows that **L1 cache is very tightly coupled memory. It is inside the chip. It is expensive to put in so not everybody does it.**

The on-chip cache is 8-K bytes. And it has both modes of write-thru and write back. The **write back cache really helps speed up burst writes** to the DRAM.<sup>2</sup>

**System Management Mode** is a totally separate address space entered via SMI that allows you to use the chip without any of the applications or the operating system having to know what's happening down below. It is a really powerful feature those people we need to really get deeply into the chip.

---

1. The ZFx86 has separate I/O and Memory address spaces, and the ZFx86 has both an ISA and PCI external bus

2. The other benefit of write back cache is that it frees up bus time for other operations, delaying writes until they are necessary.

## 32 Power Management

---

Power management goes through the entire chip. How does it get to the processor? There are two pins, SUSP#<sup>1</sup> and SUSPA# that go into the processor.

Those pins do not go off chip, so why do we mention them? Those pins have to be activated through the software from the South Bridge through the North Bridge to the processor. There's a handshaking process that takes place that brings the signals back down all the way to the South Bridge before the chip actually shuts down. It takes about 200 clock cycles. That's not that long.

**ZF will provide APM 1.2 power management support** through the Phoenix BIOS (see Interrupt 15h—APM Services in [PhoenixBIOS 4.0 Rev6 User Manual.PDF](#)).

ZF will also provide APM support as part of the ZF Board (Chip) Support Package for the VxWorks RTOS.

---

1. See file [ZFx86 Power.PDF](#) for power consumption.

## 33 Memory Address Space – SDRAM

---

In the memory space the CPU allows **4 GB** of logical address space, but the chip is physically limited to **256 MB of SDRAM**<sup>1</sup>. Intel chips map logical to physical memory space using the built in MMU.

Intel architecture provides a **protected mode feature** to allow software to be written to provide task isolation. Wind River (Tornado 3) and QNX embedded RTOS software will take advantage of this protected mode.

Intel processors have 64 KB of address space for **Input/Output** which is **outside of the memory address space**. Thus there is no conflict between the I/O Space and the DRAM space.<sup>2</sup>

---

1. You can also have PCI memory space and ISA memory space. Using the ZF-logic Memory Chip Select feature, you can add up to 64K of flash/SRAM memory which is viewed through ISA memory viewports (allowing a lot of memory to be viewed through a small ISA memory hole).

2. There are four benefits of having memory and I/O space isolated: (1) the I/O does not create holes in the DRAM space; (2) in protected mode, using the IOPL bits in the CPU, access to I/O can be restricted to processes of high enough privilege; (3) using the I/O BITMAP in the Task State Segment, *specific* I/O addresses may be made available to any task; and (4) in a sophisticated O/S like IBM OS/2, DOS sessions can claim I/O ports using the I/O bitmap. Items 2-4 are features of the protected mode operation of the Intel architecture.

## 34 Memory Address Space – ISA, PCI, ZF-logic

---

The 256 MB refers to SDRAM but does not refer to memory on the ISA bus or the PCI bus.

ISA memory can be expanded with minimum cost using the “extended digital logic”, the ZF-logic.<sup>1</sup>

---

1. See [‘Benefits of Memory Window Mapping’ on page 92](#)

**Interrupts** are one of the key elements which allows the processor to manage different tasks at the same time.

Interrupts: this is a good time to talk about the real-time operating systems because the real-time operating systems have a **very small time delay** (called latency) for processing interrupts.

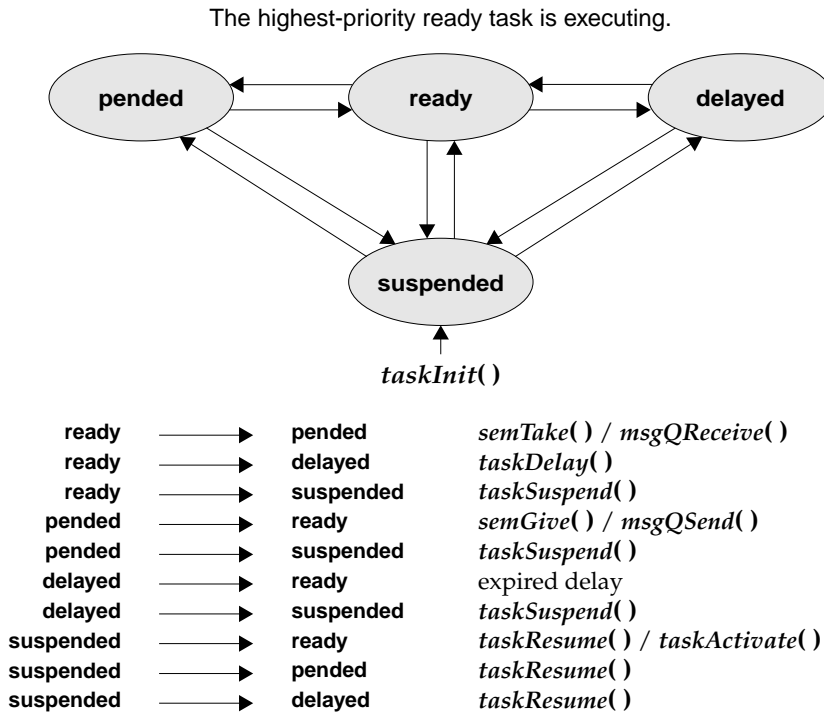
What they do is that they quickly grab control of the processor, take care of the I/O which needs to be taken care of, and then go back to normal program execution.

In a RTOS such as VxWorks, tasks can be **placed on the ready-list** based on time tick interrupts, or based on interrupt.

The **difference between an RTOS and a normal operating system** is the normal operating system really doesn't have this built-in mechanism to define which tasks are really important, and queue them based on real time events.

VxWorks 5.4  
Programmer's Guide

Figure 2-1 Task State Transitions



### 2.3.3 Wind Task Scheduling

Multitasking requires a scheduling algorithm to allocate the CPU to ready tasks. Priority-based preemptive scheduling is the default algorithm in *wind*, but you can select round-robin scheduling for your applications as well. The routines listed in Table 2-2 control task scheduling.

Table 2-2 Task Scheduler Control Routines

Call	Description
<i>kernelTimeSlice()</i>	Control round-robin scheduling.
<i>taskPrioritySet()</i>	Change the priority of a task.
<i>taskLock()</i>	Disable task rescheduling.
<i>taskUnlock()</i>	Enable task rescheduling.

Courtesy Wind River Corporation, VxWorks Programmer's Guide 5.4



Within the chip we have 15 different maskable interrupts. These are all the ISA bus interrupts which you are normally used to seeing. What happens on the ZF<sub>x</sub>86, however, is that we have a PCI bus and an ISA bus. Because we have both buses, the interrupts need to be shared between the two. The designer of the system needs to figure out which interrupts his going to use where. These maskable interrupts are used for cards on the ISA bus and cards on the PCI bus.

**NMI** (nonmaskable interrupt) this is an interrupt which can never be turned off. It is always available to the hardware. Nonmaskable interrupt allows very important events to interrupt the processor even if the processor has "turned off" interrupt because it is perhaps inside interrupt service routine for a critical section of code.

**SMI** (system management interrupt) puts a CPU into system management mode. The system management mode is a totally separate address space and interrupts space that allows you to use the device without any of the applications are headed the operating systems to know what's happening down below. It is a really powerful feature those people who need to really get deeply into the device.

**SCI** (system controller interrupt) -- generally used with ACPI (Advanced Configuration & Power Interface).

Up to eight GPIOs in the South Bridge are provided for system control. There are 8 GPIO pins on the ZFx86. The features include power management event (PME) generation. This means that any of the 8 GPIO pins set in input mode can be used to wake up the processor. That is, each GPIO pins can be programmed to generate an **SMI** or **SCI**. The Watchdog Timer can also generate RESET, NMI, SMI, or SCI. See ['Watchdog Timer' on page 99](#).

Developed by Intel, Microsoft and Toshiba and announced on January 6, 1997, **ACPI is an open industry (anyone can use it), all-encompassing, PC hardware, Operating System and peripheral device interface specification**. In other words, it specifies a certain manner in which the OS, motherboard hardware and peripheral devices (such as CD-ROMs, Hard Drives, etc.) talk to each other about power usage. Its primary goal is to enable Operating System Directed Power Management (OSPM) whereby the Operating System manages all power activities – providing power to devices only on an as-needed basis. See <http://www.teleport.com/~acpi/>

# 39 | Interrupt Vector Assignment

**Interrupt 5 is free because we only have one parallel port** onboard. 9, 10, 11, and 12 are free interrupts. The floppy, which is an internal device, picks up interrupt six.

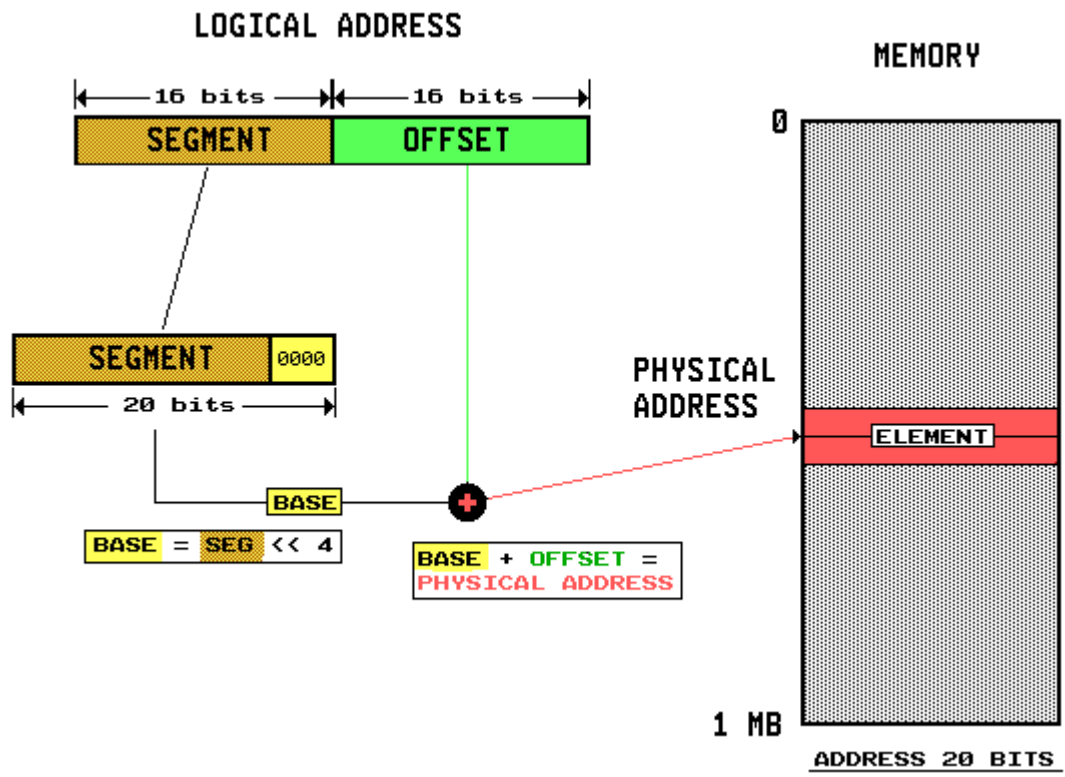
**Traditionally interrupts 9,10,11 and 12 are routed to INTA, INTB, INTC and INTD on the PCI bus.** On the PCI bus, all four interrupts go to all four of the slots. What is typically done, when a card is plugged into the first slot, yet it is a single function device it uses INTA. If it is a dual function device, say video and audio together, it uses INTA and INTB. What designers normally do is that on the first slot that been designated interrupt a goes to interrupt a. On the second slot, the designated interrupt a goes to interrupt b. On the third one, they route it to c. This allows devices which are plugged into the slots to always have high priority, because their interrupt a will be routed to 10,11, or 12. The problems come in if you try used 12 for the fourth slot (if you have a fourth slot) and you have a mouse in your system. Traditionally the mouse also occupies interrupt 12. So it's very easy to come up with a situation where you have conflicts on your interrupts. People will run into this trap over and over again. They won't know how to use these four interrupts together with interrupt 5, and they won't know which ones to put on the ISA bus in which was to put on the PCI bus.

Regarding interrupt conflicts, we have many many devices onboard. Of course, in people don't need those devices they can always turn them off.

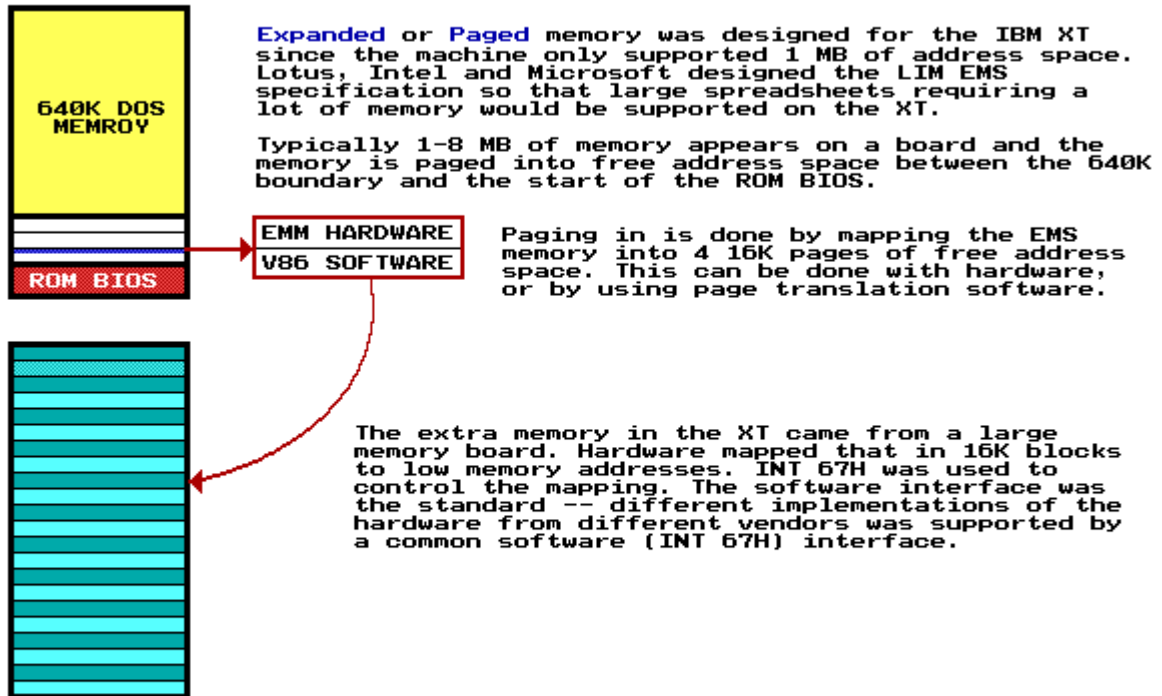
**Table 2.1: Interrupts**

INTERRUPT	USE	USE	EXTERNAL
Master 0	IRQ 0	System Timer	No
Master 1	IRQ 1	Keyboard Controller	No
Master 2	IRQ 2	Slave Interrupt Controller input	No
Master 3	IRQ 3	Secondary Serial port (shared)	Yes
Master 4	IRQ 4	Primary Serial port (shared)	Yes
Master 5	IRQ 5	User Defined	Yes
Master 6	IRQ 6	Floppy Disk	No
Master 7	IRQ 7	Parallel Port (shared)	Yes
Slave 0	IRQ 8	RTC	No
Slave 1	IRQ 9	User Defined. Can be PCI interrupt A.	Yes
Slave 2	IRQ 10	User Defined. Can be PCI interrupt B.	Yes
Slave 3	IRQ 11	User Defined. Can be PCI interrupt C	Yes
Slave 4	IRQ 12	User Defined. Can be PCI interrupt D	Yes
Slave 5	IRQ 13	Math Coprocessor	No
Slave 6	IRQ 14	Primary IDE Channel	Yes
Slave 7	IRQ 15	Secondary IDE Channel	Yes

# 40 Real Mode (DOS)

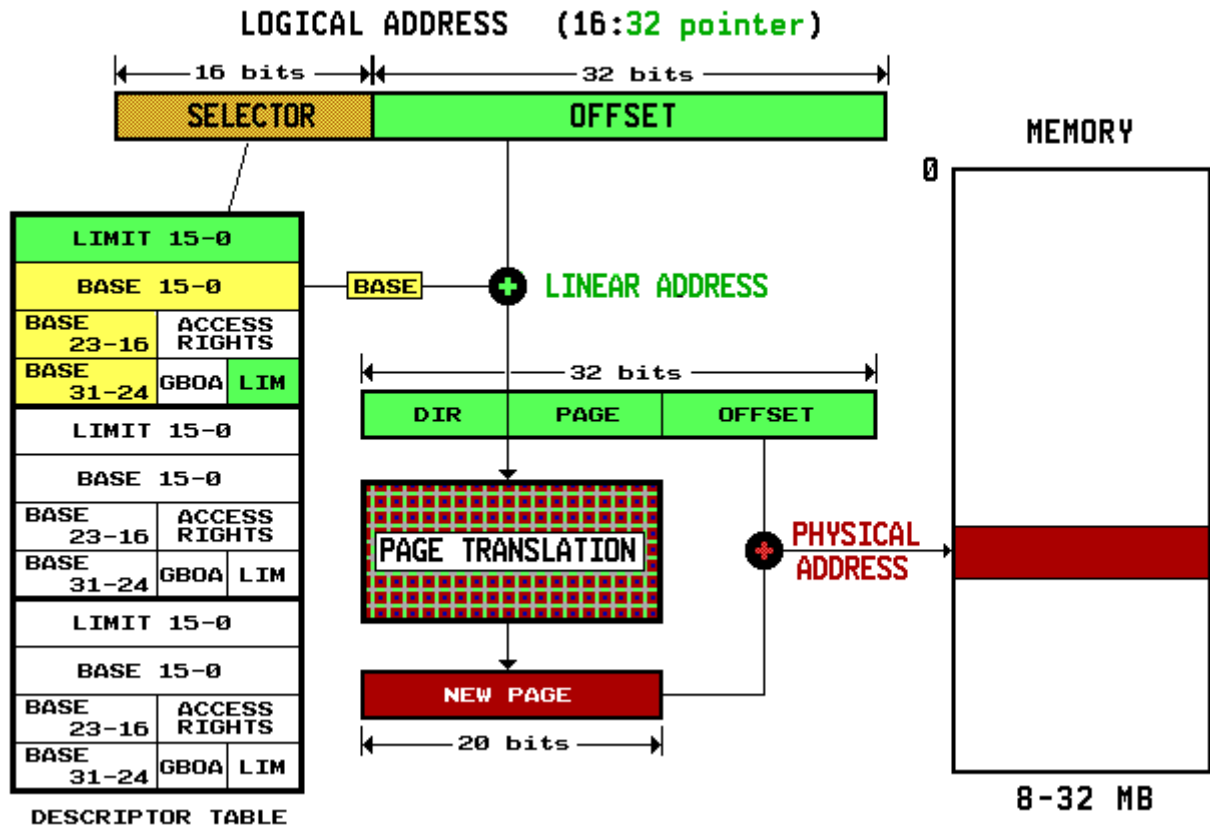


# 41 | Extended and Expanded Memory in Real Mode





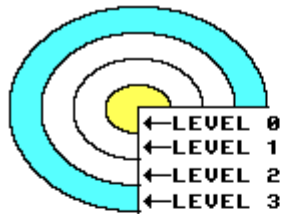
# 42 Basic Protected Mode Operation



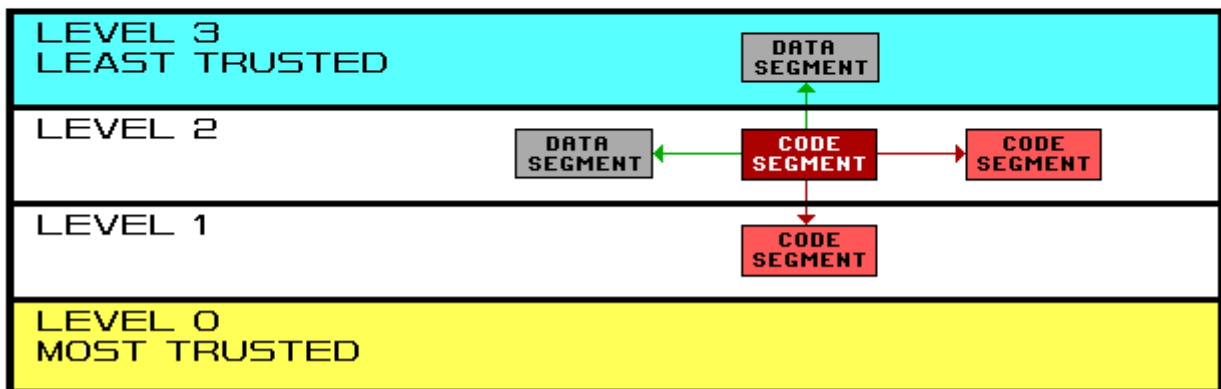
# 43 Privilege Levels – Data Access

Privilege levels: there are four different privilege levels. It's a standard x86 processor. But for those guys you need to convince the RISC processors better than this, these are the things that we pick up in legacy. A lower privileged task cannot modify memory which has been assigned by a higher privileged task or by the operating system.

## Ring Model of Protection



## RULES OF PRIVILEGE



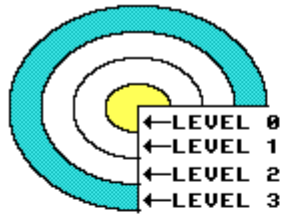
(1) A program can call to a code segment descriptor at its own privilege level, or to code segments that are more trusted. It cannot call to code segments that are less trusted.

A program can RET to a code segment at its own privilege level, or to code segments which are less trusted.

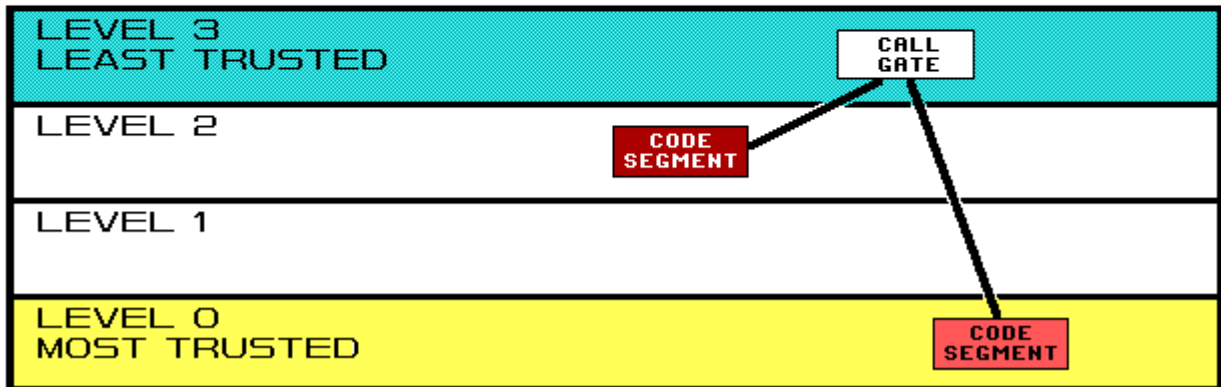
(2) A program can access data at its own privilege level, or data which is less privileged. It cannot access data which is more privileged.

(3) There are two bits in the flag word called the IOPL (input/output privilege level) which establish what degree of privilege is required before a program can execute any IN or OUT instructions.

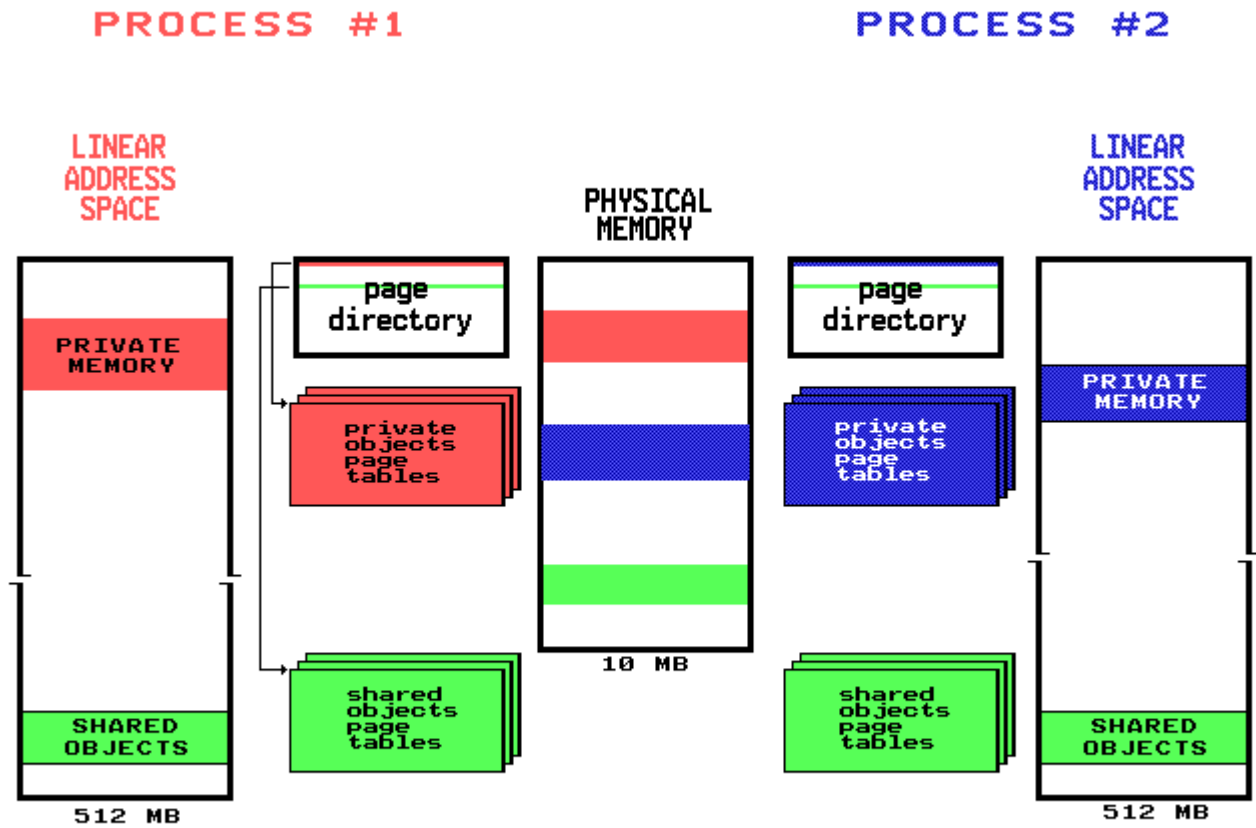
# 44 Privilege Levels – Code Access



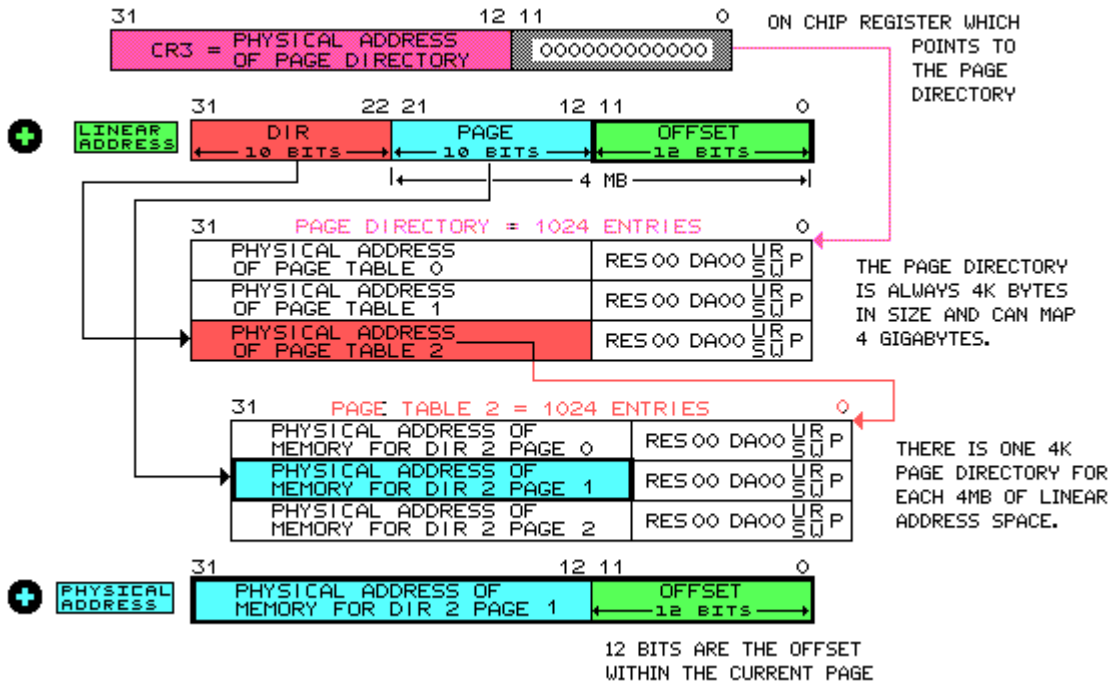
## TRANSFER THROUGH A CALL GATE



# 45 MMU – Page Level Protection

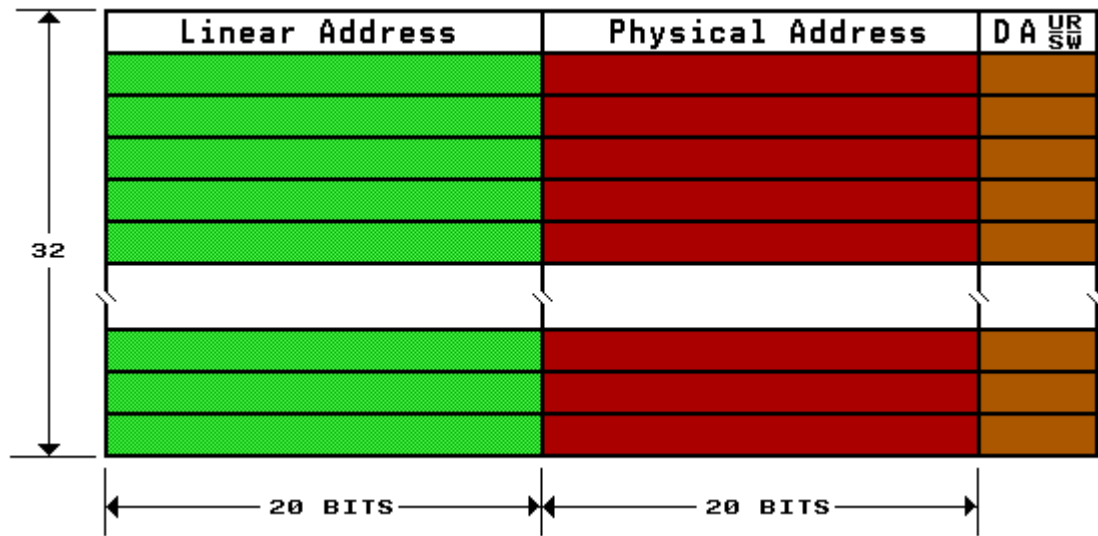


# 46 MMU – Page Directory and Page Tables





# 47 MMU – Translation Look-Aside Buffer



# 48 IOPL and SMM Protection

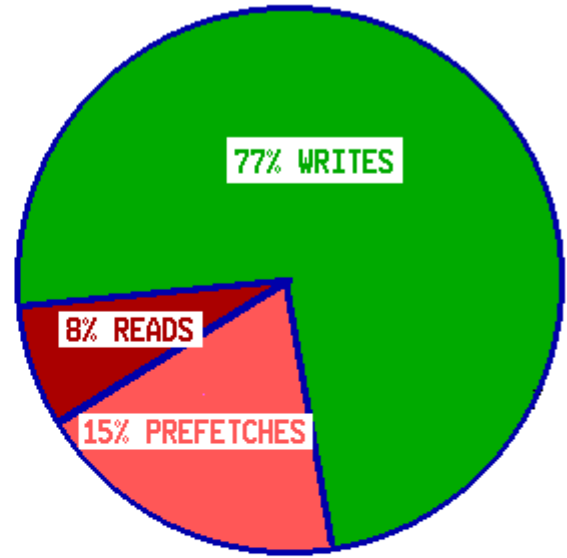
---

IOPL Protection

SMM Protection



386DX CPU BUS CYCLE MIX



80486 CPU BUS CYCLE MIX

## 50 Integrated FPU

---

The floating point unit allows you to process information much faster. Some of the newer operating systems require the floating point unit.

## **VALIDATION OF x86 ARCHITECTURE:**

LONG WORD INSTRUCTION PROCESSOR SIMULATES x86 PROCESSOR.

ZFx86 NATIVE x86 - CROSS LICENSED W/ INTEL  
PERFORMANCE IMPACT DUE TO "CODE MORPHING" SW  
HAD TO "RE-INVENT THE WHEEL" AND DEVELOP A NEW  
PROCESSOR ARCHITECTURE

## **LIMITED INTEGRATION**

ONLY HAS PCI, SDRAM AND ROM INTERFACES  
DOES NOT HAVE FLOPPY, IDE, USB, SERIAL, PARALLEL,  
GPIO, PWM, ISA, I2C, REALTIME CLOCK, IRQ, DMA,  
MOUSE, KEYBOARD, WATCH DOG TIMER  
DOES NOT HAVE VARIABLE WIDTH DRAM BUS,  
THEREFORE, MUST USE 32MB DRAM.  
CODE MORPHING CODE USES 8-16MB DRAM.  
DOES NOT HAVE BUILT IN ROM.

## **POWER SAVINGS MODES**

POWER SPECS STATED HAVE "POWER MANAGEMENT  
ENABLED" THIS MEANS THAT PEAK POWER  
CONSUMPTION IS AT LEAST 2X THE CHART.  
ZFx86 WITHOUT POWER MANAGEMENT IS ABOUT 1 W.  
ZFx86 HAS SAME POWER MANAGEMENT STATES WITH  
THE ADDITION OF MANY DIFFERENT AND DISTINCT  
WAKE-UP INTERFACES



### **SERIAL ROM INTERFACE**

ZFx86 HAS PATENT PENDING ON ZTAG INTERFACE.  
MAY COVER CRUSOE SERIAL ROM INTERFACE.

### **CERAMIC PACKAGE**

ZFx86 HAS PLASTIC (LESS EXPENSIVE) PACKAGE

### **COMPETITION AIMED AT PENTIUM SPACE**

CHASING MEGAHERTZ AND DESKTOP PROCESSORS  
AGAINST BILLION DOLLAR COMPETITORS

### **SW INTEGRATION MISSING**

DOES NOT INCLUDE BIOS OR REAL TIME OS.

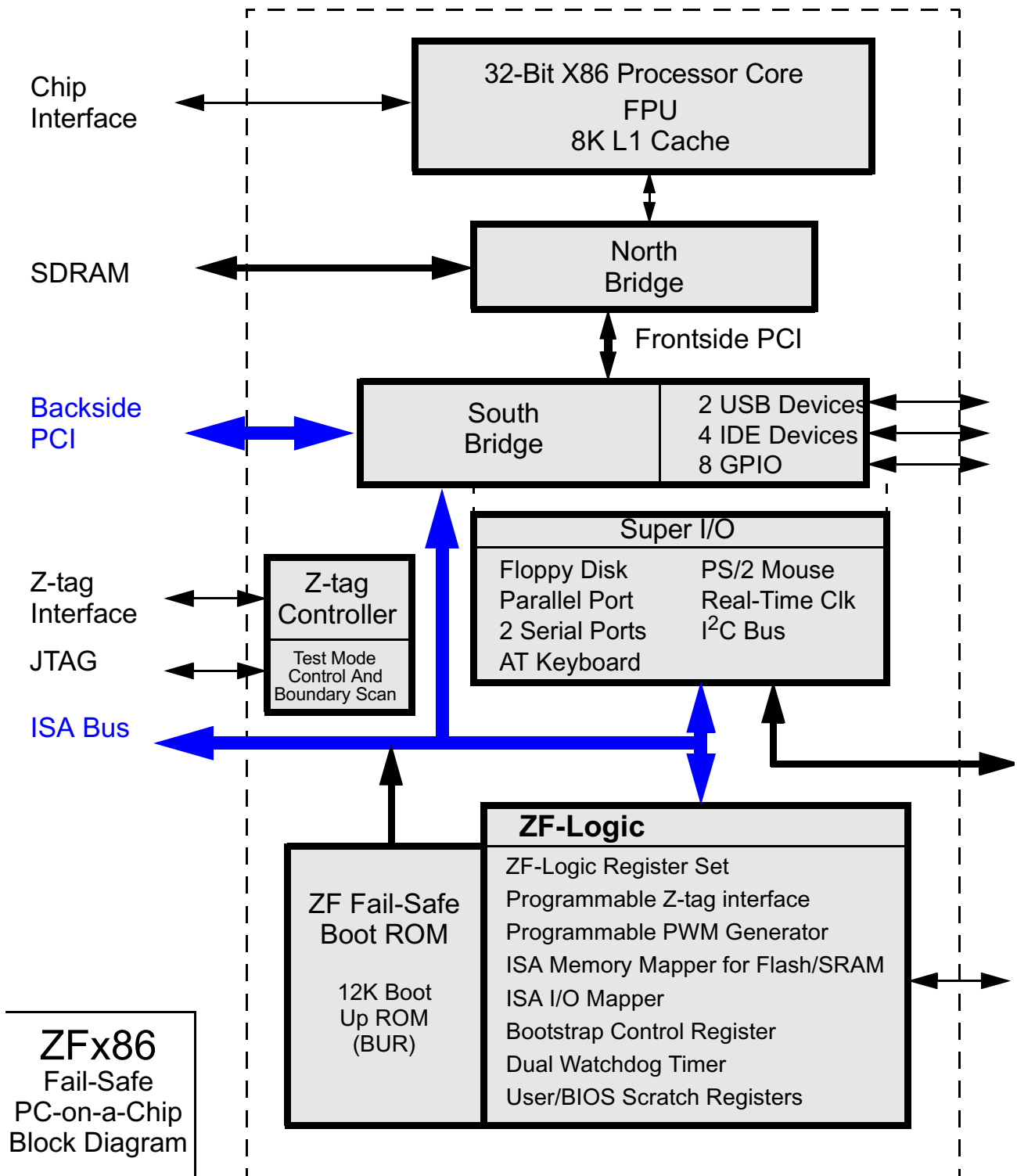
### **SYSTEM COST MUCH HIGHER**

LACK OF INTEGRATION REQUIRES ADDITIONAL CHIPS  
WIDE DRAM BUS FORCES A FOUR DRAM BUS  
CONFIGURATION

# ZFx86 Training Book

## Chapter 3 - North Bridge

# 54 ZFx86 Block Diagram



## 55 North Bridge Features

---

- CPU single cycle and burst bus transactions support
- Cache coherency support
- Support for SMM bus cycles
- Memory Controller to support Synchronous DRAM (SDRAM). Memory can be configured as 16 or 32 bits wide.
- Support for up to four banks of SDRAM and 256 Mbytes of memory space
- CPU bus to PCI bus bridge with PCI arbiter. Support for **three** external masters and one internal master. Any on chip or off chip master must connect via this interface.
- External PCI bus mastership. External bus mastership of System Controller internal bus. Mastership allows access to system controller memory devices.
- SDRAM Write Buffer – 32 Bytes
- CPU to PCI Write Buffer – 32 Bytes
- PCI Write Buffer – 16 Bytes
- PCI Read Pre-fetch Buffer – Dual 16 Bytes
- Support for Power Management signals from the South Bridge

## 56 North Bridge Features Q&A

---

1] What is “**North Bridge CPU single cycle and burst bus transactions support**”? What is a burst cycle and how and why does the North Bridge get involved?

a] \_\_\_\_\_  
\_\_\_\_\_

2] What is **cache coherency support** and why would the North Bridge need to support it? Is this a unique ZFx86 feature?

a] \_\_\_\_\_  
\_\_\_\_\_

3] What are **SMM bus cycles** and why would the North Bridge need to support them?

a] \_\_\_\_\_  
\_\_\_\_\_

4] Why is there a benefit if the chip supports **memory which is either 16 or 32-bits wide**? Why is there a benefit of having **4 banks** each having a maximum size of 64 MB rather than 1 bank with a maximum size of 256 MB?

a] \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

5] Why do we need a **CPU bus to PCI bus bridge**? What is a CPU bus and what is a PCI bus? What is a bridge? What is the difference between a **North Bridge and a South Bridge**? Why do we discuss **arbitration** and PCI bridges at the same time, are they related? Is this unique to the ZFx86?

a] \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

## 57 North Bridge Features Q&A (Continued)

---

6] What is **support for three external masters and one internal master**. Who could be a master? What is an internal master and why can you only have one? What is an external master (external to what)? Why can you only have three? What would be a typical external master and why would one want to have one?

a] \_\_\_\_\_  
\_\_\_\_\_

7] What is the benefit of all of these buffers? If they are so important, then doesn't everyone have them?

- SDRAM Write Buffer – 32 Bytes
- CPU to PCI Write Buffer – 32 Bytes
- PCI Write Buffer – 16 Bytes
- PCI Read Pre-fetch Buffer – Dual 16 Bytes

a] \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

8] What is the relationship between **Power Management** and the South Bridge? Why does the North Bridge need to support power management signals in the South Bridge? Where do they come from, where do they go, and how are the signals turned on and off? Is this any different than the way that other contemporary chips work?

a] \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

[ ] Check Your Answers on Page [page 66](#).



## 58 North Bridge Overview

---

The North Bridge components was traditionally a Pentium class North Bridge; many of the features that you will find in the North Bridge are all Pentium class features. One of the things which is unique about our design is that the processor is fast enough so that it competes with some of the lower end Pentium's in terms of processor power.

When you go to the manual in the North Bridge a lot of these things are pointed out: cache coherency, CPU write-thru and write back cache support etc.

We do concurrent PCI and DRAM access.

We now have chips at 66 MHz system clock which means that the DRAM interface for the chip is running at 66 MHz. That is a very high-performance configuration. The system performance is almost double having a 66 MHz system clock and a 133 MHz processor vs. having a 33 MHz system clock and a 133 MHz processor. The processor just starves to death with no data.

## 59 North Bridge SDRAM Controller

---

There are lots of things in the North Bridge. But the **North Bridge has two major tasks: the first one is an SDRAM controller.** The SDRAM controller manages the DRAM facilities are onboard.

SDRAM rather than EDO/Fast Page RAM

We have chosen an SDRAM interface which is much faster than the normal EDO. or fast page mode interfaces. Also, EDO and fast page mode are quickly going away and you can't buy them.<sup>1</sup>

We have up to four banks and up to 256 MB of memory, so that the really want to have a lot of memory in your system you are welcome to do that.

SDRAM vs. EDO or fast page mode memory. SDRAM is a very fast DRAM. Essentially once we've pumped up and put out two clock pulses it will provide data on every clock. That is much faster, almost twice as fast, as EDO or fast page mode memory.

It is important we provide support for 16 Mb, 64 Mb and 128 Mb (bit) devices with addresses to A11 (12 address lines). You can mix DRAM sizes and widths on different banks.

---

1. The SDRAM interface is a good feature to have. Many of the competitors still use EDO memory.

The Elan 410/410 is still stuck with older DRAM interfaces. The new AMD processor does have the SDRAM interface, but it is the only one. Many RISC processors still use EDO/fast page mode memory. So this advantage is important.

## 60 North Bridge SDRAM Controller (Continued)

---

### 16 or 32-bit wide DRAM Bus

One key feature that we have which allows us to have a \$30 or \$40 cost advantage over the competition is the ability to go with either a 16-bit wide or a 32-bit wide DRAM bus. Why is this important? Because most of the devices which are sold are sold in configurations which are only 16-bit wide

When you power up the system for the first time with our chip all you need is a single DRAM chip 16-bits wide. When you look that all the competitors, especially the MediaGX, the Pentium class processors, ST microelectronics processor, the industrial PC, the consumer PC, they all have 64-bit wide DRAM busses. The 64-bit means that as a minimum you need four DRAM chips to start up the system. That tend to \$13 a chip, you have three extra chips that you need. That gives us a 30 to \$40 price advantage for anybody who wants a really small embedded system.

## 61 North Bridge PCI

---

The PCI features of the North Bridge: the North Bridge controls who gets on the first. There's bus mastering which occurs. This essentially means that any device which is on the PCI bus can take control of the entire system and grab memory by itself. This is arbitrated by the North Bridge.

You have concurrent PCI and DRAM access from the processor, allowing the processor to write to the PCI and the DRAM at the same time. This is a very powerful feature.

The North Bridge has CPU single cycle and burst bus transaction support. That means that you can ask for one word at a time, or a string of words at a time.

## 62 North Bridge Cache Management

---

Cache management: when you have a cache on chip the data inside starts getting old. There are times when the North Bridge decides the data that is inside the cache needs to be flushed back to main memory because some device is asking for that memory location (that has been updated in the cache already).

## 63 | Bus Arbitration on the PCI Bus

---

Bus arbitration: essentially the North Bridge is the guy who decides who gets the PCI bus. The North Bridge is the one which will support up to three extra masters and two internal masters: the CPU and the South Bridge. So we have up to five devices that can master the PCI bus: three of them are external and two of them are internal. When you look at the competitors, AMD has the Elan 520 which allows five bus masters: this is one area where they have a couple more features than we do. The normal answer that I would provide anyone who is questioning me on this area is that if you have a processor with this kind of processing power, to put by PCI bus masters on it is probably overkill anyway.<sup>1</sup>

---

1. However, you can always add an external chip which allows you to piggyback bus masters. Each chip allows you four more masters, but takes up one master. So each chip gives you a net gain of three masters.



## 64 PCI vs. ISA bus mastering

We do not support ISA bus mastering. Typically bus mastering is something which issues to make the system go very fast. To make things happen quickly. Since we do support PCI bus mastering, if any customer has a concern about data availability or getting data to the processor issue just use the PCI. When you look at some of the differences, the clock is 8 MHz on the one hand 33 MHz on the other.

Item	ISA	PCI	Comment
<b>Clock</b>	<b>8 MHz</b>	<b>33 MHz</b>	<b>factor of 4</b>
<b>Data width</b>	<b>16 bits</b>	<b>32 bits</b>	<b>factor of 2</b>
<b>Bus Mastering</b>	<b>No</b>	<b>Yes</b>	<b>Complexity of adding ISA bus mastering to North bridge controller was not warranted.</b>

Although ISA bus mastering is not supported, DMA is supported on the ISA bus. So you can have direct memory access from the ISA bus but it will involve the CPU. We have both an 8-bit and 16-bit DMA transfer available to/from the ISA bus.

The complexity that we would have had to add to the chip do support ISA as well as PCI bus mastering was not worth the trade-off.

## 65 | The Role of the NB in Power Management

---

The other thing that the North Bridge does is support the power management signals from South Bridge.

# Answers to Questions

<[page 56](#)>

1] What is “**North Bridge CPU single cycle and burst bus transactions support**”? What is a burst cycle and how and why does the North Bridge get involved?

a] \_\_\_\_\_  
\_\_\_\_\_

2] What is **cache coherency support** and why would the North Bridge need to support it? Is this a unique ZFx86 feature?

a] \_\_\_\_\_  
\_\_\_\_\_

3] What are **SMM bus cycles** and why would the North Bridge need to support them?

a] \_\_\_\_\_  
\_\_\_\_\_

4] Why is there a benefit if the chip supports **memory which is either 16 or 32-bits wide**? Why is there a benefit of having **4 banks** each having a maximum size of 64 MB rather than 1 bank with a maximum size of 256 MB?

a] \_\_\_\_\_  
\_\_\_\_\_

5] Why do we need a **CPU bus to PCI bus bridge**? What is a CPU bus and what is a PCI bus? What is a bridge? What is the difference between a **North Bridge and a South Bridge**? Why do we discuss **arbitration** and PCI bridges at the same time, are they related? Is this unique to the ZFx86?

a] \_\_\_\_\_  
\_\_\_\_\_

6] What is **support for three external masters and one internal master**. Who could be a master? What is an internal master and why can you only have one? What is an external master (external to what)? Why can you only have three? What would be a typical external master and why would one want to have one?

a] \_\_\_\_\_  
\_\_\_\_\_

7] What is the benefit of all of these buffers? If they are so important, then doesn't everyone have them?

- SDRAM Write Buffer – 32 Bytes
- CPU to PCI Write Buffer – 32 Bytes
- PCI Write Buffer – 16 Bytes
- PCI Read Pre-fetch Buffer – Dual 16 Bytes

a] \_\_\_\_\_  
\_\_\_\_\_

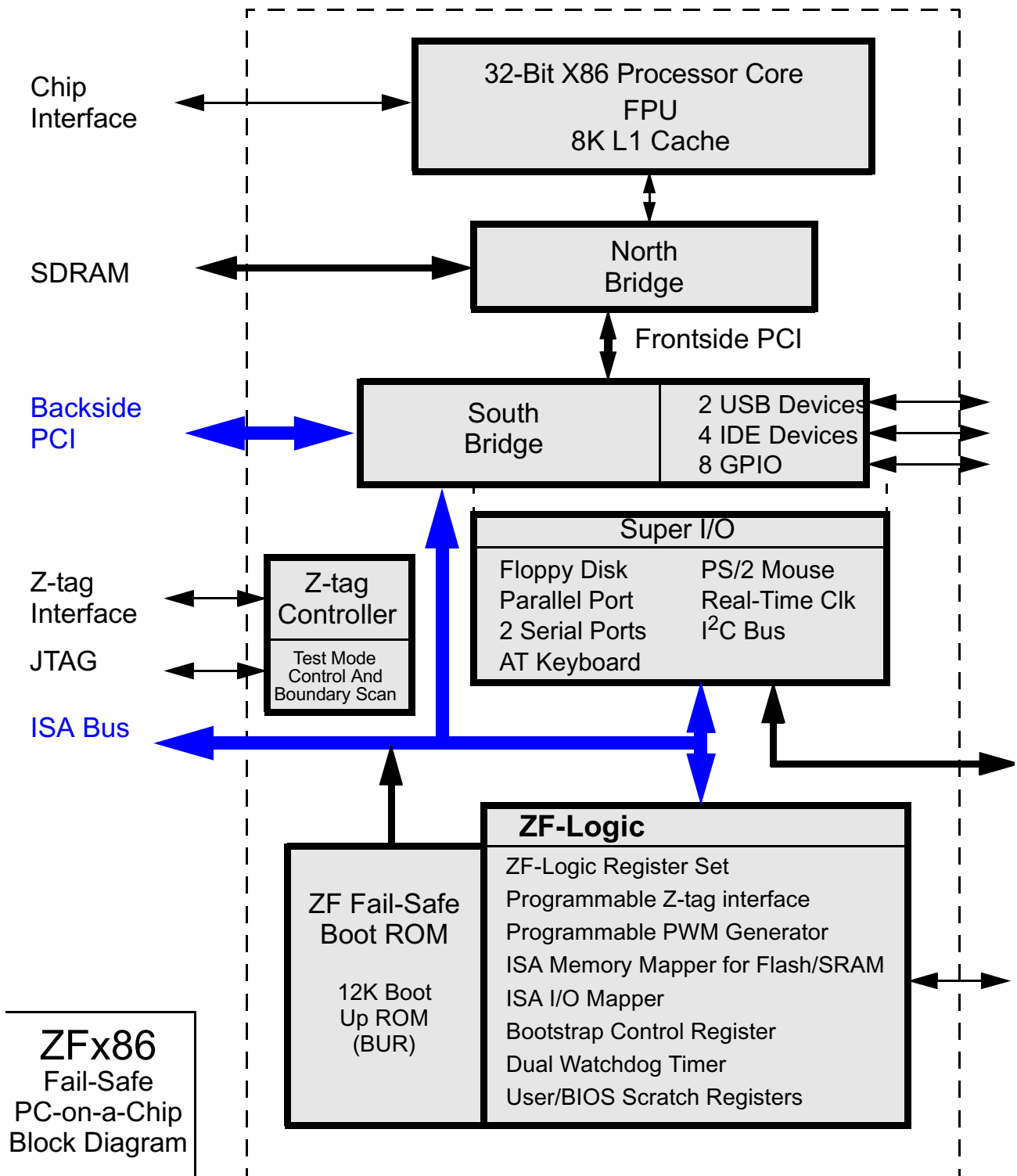
8] What is the relationship between **Power Management** and the South Bridge? Why does the North Bridge need to support power management signals in the South Bridge? Where do they come from, where do they go, and how are the signals turned on and off? Is this any different than the way that other contemporary chips work?

a] \_\_\_\_\_  
\_\_\_\_\_

# ZFx86 Training Book

## Chapter 4 - South Bridge

# 69 ZFx86 Block Diagram





## 70 Definition of Functional Blocks

---

Looking to block diagram, the South Bridge is another very standard Pentium class device. Some devices are internal to the South Bridge. They are connected directly on the front side PCI bus. These are the USB devices, the IDE devices, and GPIO. All of these come on chip. In addition to that the South Bridge controls what is called the Super I/O.

Functional blocks

Here is a list of things that the South Bridge does.

- Front Side PCI Interface
- Back Side PCI Interface
- Bus Mastering IDE Controller
- Universal Serial Bus
- Integrated Super I/O
- ACCESS.bus interface
- AT Compatibility
- ISA Interface
- Power Management
- GPIOs

# 71 | Bus Comparison

Bus Type	Speed (MB/s)	Distance	Noise	Device Complexity
Access <sup>a</sup>	0.1875	12 inches	Low	Simple
Serial <sup>b</sup>	0.0115	100 feet	Med/High	Medium
Parallel		10 feet	Low	Medium
ISA <sup>c</sup>	4	12 inches	Low	High
USB <sup>d</sup>	1,2,4	5 feet	High	Medium
PCI <sup>e</sup>	132	4 inches	Low	High
IDE <sup>f</sup>	4	12 inches	Low	High
GPIO	1	12 inches	Low	Simple
IrDA <sup>g</sup>	0.115 - 4.0			
Z-Tag	1.5			Simple

a. For I2C Bus, see <http://www-us.semiconductors.com/i2c/>

b. See [‘Serial Port’ on page 76](#)

c. See ISA System Architecture, by Don Anderson and Tom Shanley (MindShare) (ISBN 0-201-40996-8)

d. See <http://usb.org/>. See also [‘USB.ORG Website’ on page 74](#).

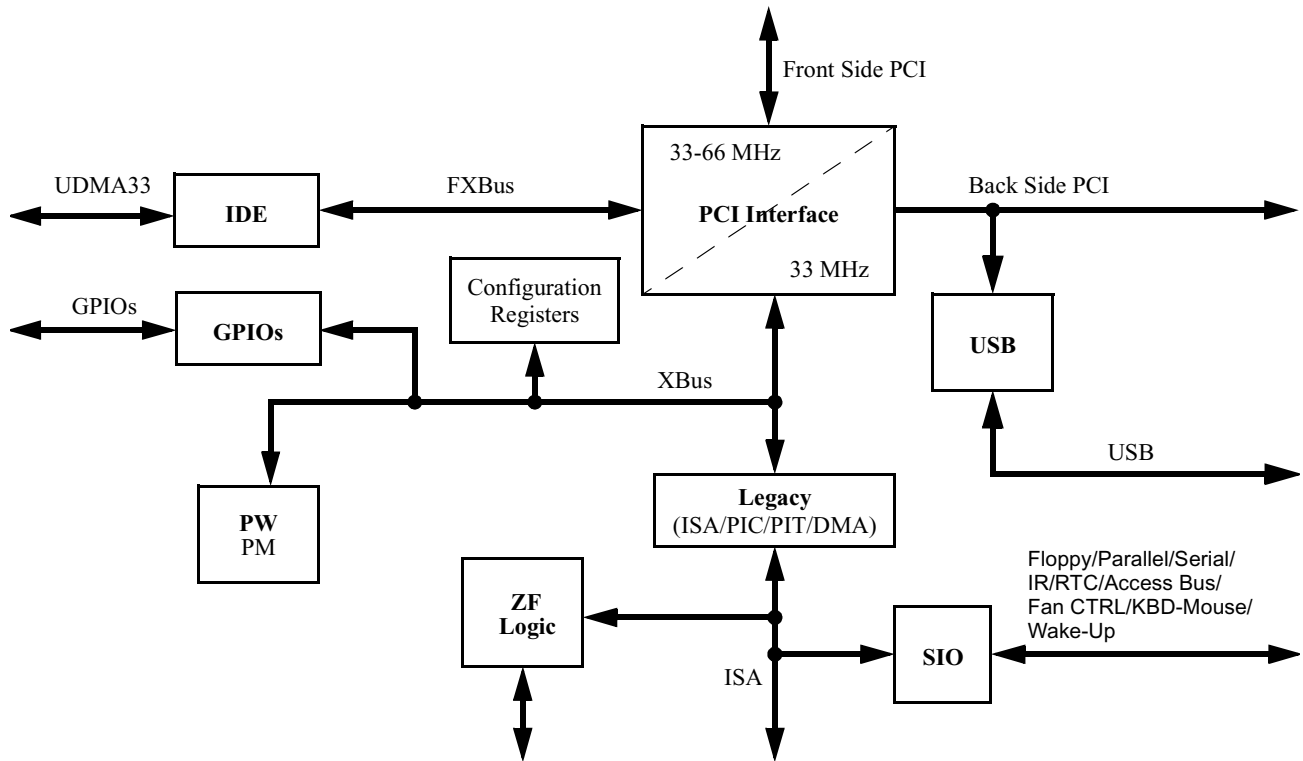
e. See PCI Hardware and Software Architecture & Design, 4th Edition, Solari & Willse, Annabooks, ISBN 092939259-0. See also the PCI SIG on the web at <http://www.pcisig.com/index.php3?t2=1>

f. For Ultra DMA Ultra DMA Implementation Guide, <http://www.wdc.com/products/drives/drivers-ed/udmatp.html>

g. For IrDA and Linux, see <http://www.cs.uit.no/linux-irda/>. Some IrDA Data Sheets appear on the ZF86 IDS CD. The SuperI/O incorporates an Infrared Communication Port that supports FIR, MIR, HP-SIR, Sharp-IR, and Consumer Electronics-IR. The IrDA data rate is up to 115.2 Kbps (SIR), Data rate of 1.152 Mbps (MIR), Data rate of 4.0 Mbps (FIR). See [‘Infrared Communication Port’ on page 77](#).

## 72 Front Side Vs Back Side PCI

It is possible to clock the front side of the South Bridge at one speed and the Back Side side of the chip at another speed.




## 73 Bus Mastering IDE Controller

---

The IDE controller is very powerful. It supports all the types of IDE transfers that exist. It supports the PIO modes 0,1,2,3, Ultra DMA, ATA format compatible. If you get the question "does the chip do it", the answer is yes.

- One channel with support for up to two IDE devices
- Second IDE channel for two more devices off GPIO
- Independent timing for master and slave devices
- PCI bus master burst reads and writes
- Ultra DMA (ATA-4) support
- Multiword DMA support
- Programmed I/O (PIO) Modes 0-4 support

CHANNEL
PRESS
DEVELOPERS
MEMBERS
PRODUCTS



**Shopping for USB**  
[USB Features](#)  
[USB FAQ](#)  
[About USB-IF, Inc.](#)


**Vote here!**

Which of the following is true of Hi-Speed USB 2.0?

a) 40X faster than the original USB

b) Fully compatible with the original USB

c) Certified products bear this logo:




d) Awarded CNET's "Best Emerging Technology Award"


e) All of the Above

[View Results](#)


Vote!


**Featured Product**


Addonics AEED25U, Pocket USB 2.0 Hard Drive 



**welcome**  
*from the creators of USB*

**Hi-Speed USB 2.0 Gears Up for Broad Deployment in 2002**  
 Press, analysts invited to hear how milestones, announcements through end of year are laying the foundation  
[\(pdf, 94k\)](#) 


**USB Implementers Forum Signals Broad Development of Hi-Speed USB 2.0 at Intel Developer Forum**  
 First-time demonstrations, announcements and other displays of upcoming broad deployment of USB 2.0  
[\(pdf, 93k\)](#) 

**USB Implementers Forum Drives Bus Into the Fast Lane**  
 Microsoft Demonstrates USB 2.0 Drivers for Windows XP at USB 2.0 Developer Conference.  
[\(pdf, 13k\)](#) 


**About USB**  
 Universal Serial Bus (USB) connects more than computers and peripherals. It has the power to connect you with a whole new world of PC experiences.


**PC MAGAZINE** THE AWARD GOES TO...

**Winner: Hi-Speed USB 2.0 for Technical Excellence**

PC Magazine has bestowed the Eighteenth Annual Award for Technical Excellence to **USB 2.0** in the Specifications category. This award is given to innovators who have found new solutions to old problems. USB 2.0 was selected for this honor because of the common interface it provides and the high data transfer rate. 

**CNET's 'Best Emerging Technology' award goes to USB 2.0 at PC Expo**

In the area of Best Emerging Technology, CNET's award goes to USB 2.0. This award does not go to any one company but to all of the industry pioneers that recognize the value of this new standard. The factors that make USB 2.0 so promising are its speed, its backward compatibility with USB 1.1, and its broad industry support. 

...USB 2.0! 

### What's inside the Super I/O?

- PC98 and ACPI Compliant
- Floppy Disk Controller (FDC)
- Parallel Port
- Serial Ports 1 and 2
- Infrared Communication Port
- Keyboard and Mouse Controller (KBC)
- System Wake-Up Control (SWC)
- Real-Time Clock

Access bus. See text on [page 80](#).

Parallel Port (EPP, ECC, IEEE 1284). Once again, does the chip do it? Yes! See text on [page 78](#).

- It does EPP.
- It does ECC.
- It does IEEE 1284.
- It is fully compatible.



What you will find is the chip does what has traditionally been done with the serial port, plus. And with this chip, there's almost always a plus. For example, what is the traditional serial ports rate? 115K baud. How fast does this one go? 1.5 Mb. We always to what the traditional thing does plus. Does the chip do it? Yes, it's in there!

- Software compatible with the 16550A and the 16450
- Shadow register support for write-only bit monitoring
- UART data rates up to 1.5 Mbps

The Infrared Comm Port contains the following:

- Data rate of up to 115.2 Kbps (SIR)
- Data rate of 1.152 Mbps (MIR)
- Data rate of 4.0 Mbps (FIR)
- Selectable internal or external modulation/demodulation (Sharp-IR)
- Consumer-IR (TV-Remote) mode
- Software compatible with the 16550A and the 16450
- Shadow register support for write-only bit monitoring
- HP-SIR
- ASK-IR option of SHARP-IR
- DASK-IR option of SHARP-IR
- Consumer Remote Control supports RC-5, RC-6, NEC, RCA and RECS 80
- Non-standard DMA support - 1 or 2 channels

Parallel Port (EPP, ECC, IEEE 1284). Once again, does the chip do it? Yes! It does EPP. It does ECC. It does IEEE 1284. It is fully compatible.

- Software or hardware control
- Enhanced Parallel Port (EPP) compatible with new version EPP 1.9 and IEEE 1284 compliant
- EPP support for version EPP 1.7 of the Xircom specification
- EPP support as mode 4 of the Extended Capabilities Port (ECP)
- IEEE 1284 compliant ECP, including level 2
- Selection of internal pull-up or pull-down resistor for Paper End (PE) pin
- PCI bus utilization reduction by supporting a demand DMA mode mechanism and a DMA fairness mechanism
- Protection circuit that prevents damage to the parallel port when a printer connected to it powers up or is operated at high voltages, even if the device is in power-down
- Output buffers that can sink and source 14 mA

System wake-up control. There are all types of real-time clock alarms. This device can be awakened in so many ways that once again the answer (to the question does the chip do it) is yes! You can wake up on the modem, on the keyboard, on right clicks, on left clicks, on double clicks, on general-purpose events (GPIO inputs) -- -- there are all kinds of different ways to wake up the chip after you put it to sleep.

- Power-up request upon detection of Keyboard, Mouse, RI1, RI2, RING, PME1 and PME2 activity, as follows:
  - Preprogrammed Keyboard or Mouse sequence
  - External modem ring on serial ports
  - Ring pulse or pulse train on the RING input
  - General-purpose events, PME1 and PME2
- Battery-backed wake-up setup
- Power-fail recovery support

## 80 Access (I<sup>2</sup>C) Bus

---

What is an access bus? It's a very simple interface -- it's two wires that allows you to do multi-drop. Multi-drop allows you to put more than one device on the chain. The access bus is compatible with the Philips I<sup>2</sup>C bus or Intel's System Management Bus (SMB). It does what these buses do and more. It runs up to 1.5 Mbps.

Real Time Clock contains the following:

- Accurate timekeeping and calendar management
- Alarm at a predetermined time and/or date
- Three programmable interrupt sources
- Valid timekeeping during power-down, by utilizing external battery backup
- 242 bytes of battery-backed RAM
- RAM lock schemes to protect its content
- Internal oscillator circuit (the crystal itself is off-chip), or external clock supply for the 32.768KHz clock
- A century counter
- PnP support
- Relocatable index and data registers
- Module access enable/disable option
- Host interrupt enable/disable option
- Additional low-power features such as:
  - Automatic switching from battery to VCC\_IO
  - Internal power monitoring on the VRT bit
  - Oscillator disabling to save battery during storage
- Software compatible with the DS1287 and MC146818

## 82 Real Time Clock Alarms

---

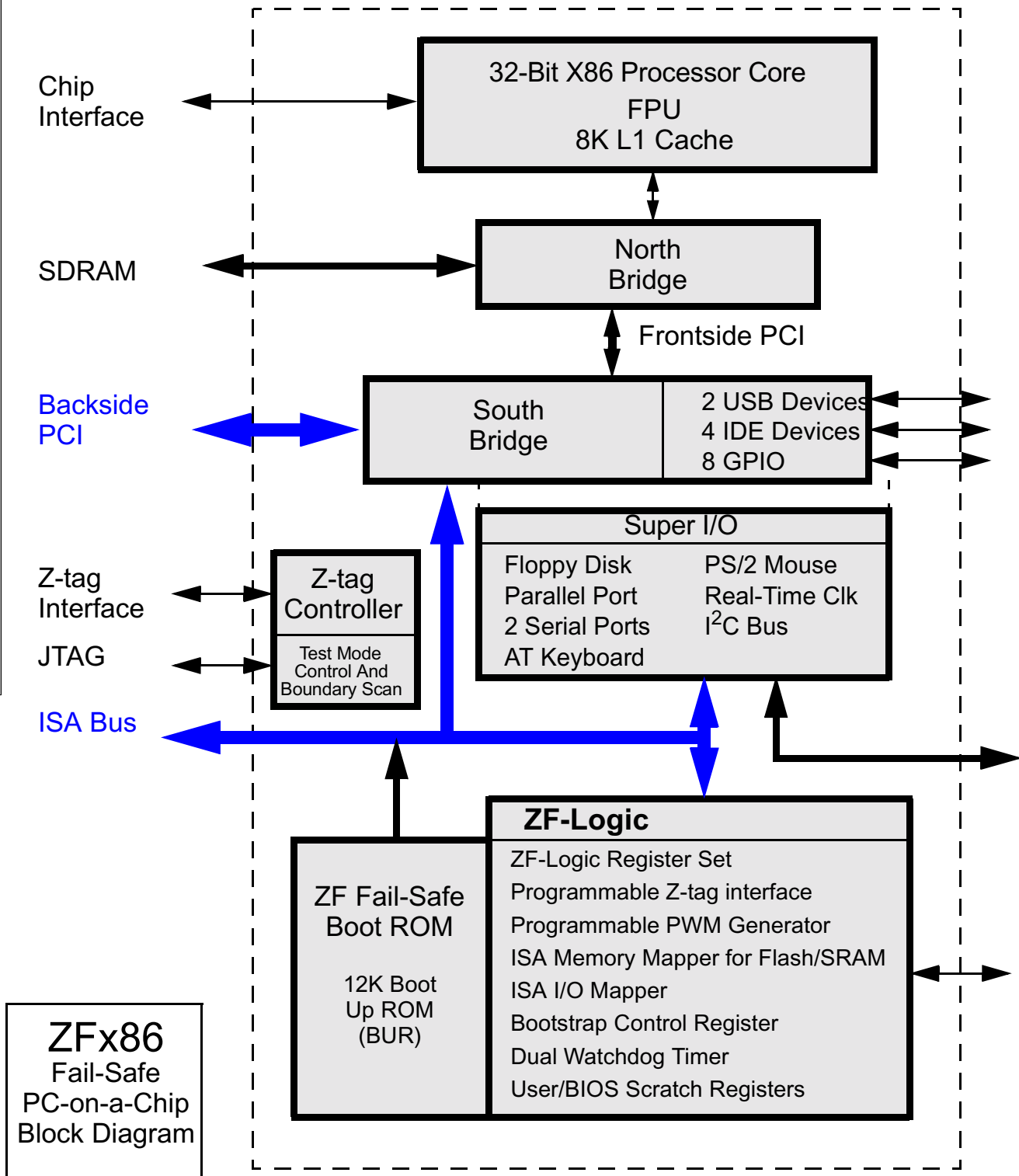
Real-time clock alarms: the timekeeping function used generate an alarm when the current time reaches a stored alarm time. Essentially, you wake up the chip based on an alarm instead of an external event, or you use the alarm to trigger various events

# ZFx86 Training Book

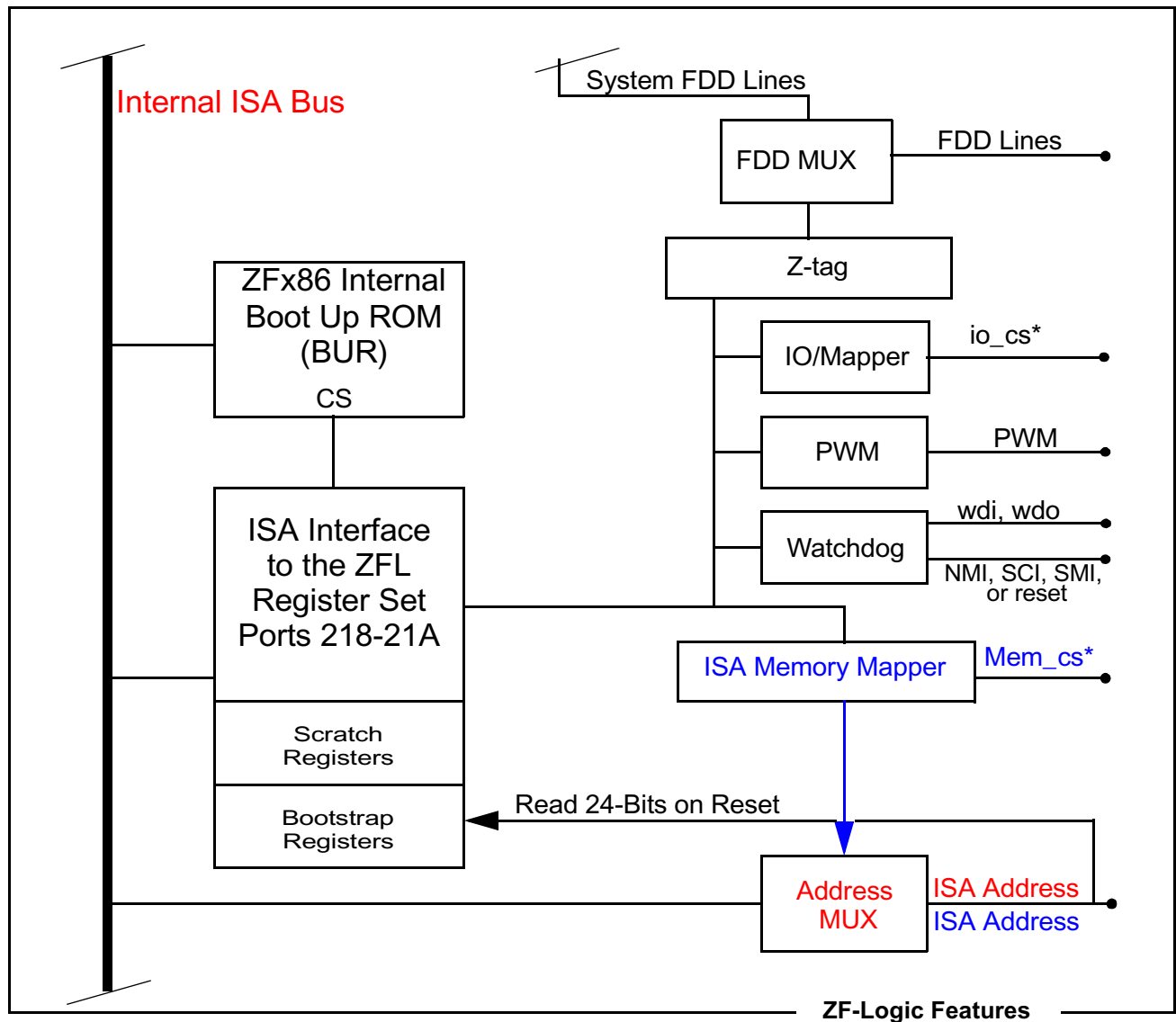
## Chapter 5 - ZF-Logic



# 84 ZFx86 Block Diagram



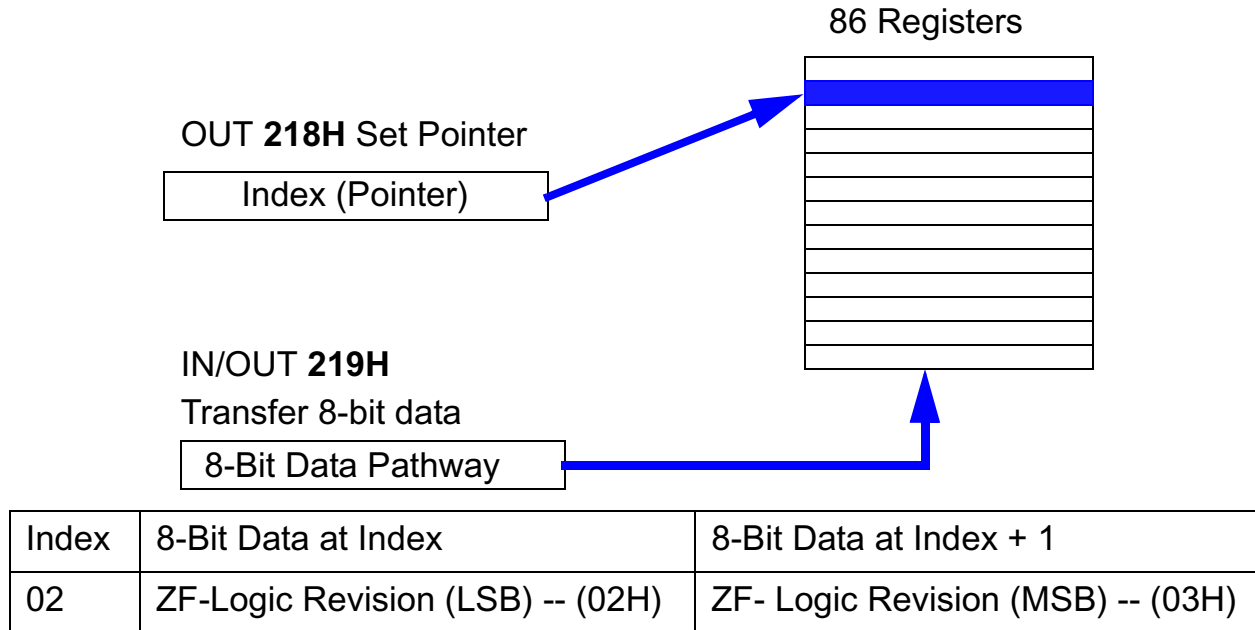
# 85 ZF-Logic Block Diagram



Note: The features of the ZFL include:

- ZFL Register Set in ISA I/O Space
- Programmable PWM generator
- Programmable Watchdog timer
- ISA Memory Mapper for Flash/SRAM
- ISA I/O Mapper General Purpose Chip Select (GPCS)
- Programmable Z-tag Interface
- Bootstrap Register (DIP switches/Pull-Ups) External Control of Boot Process
- User and BIOS Scratch Registers

# 86 ZF-Logic Register Space Access (8-Bit)



- ZF Logic is Controlled or Accessed through about 86 8-bit registers.
- Four ISA I/O Addresses are used provide Access to The Registers

## QUESTIONS:

1] Why do we have 86+ Registers in the ZF Logic?

a] \_\_\_\_\_  
\_\_\_\_\_

2] Why do we use 4 (rather than 86+) ISA Addresses?

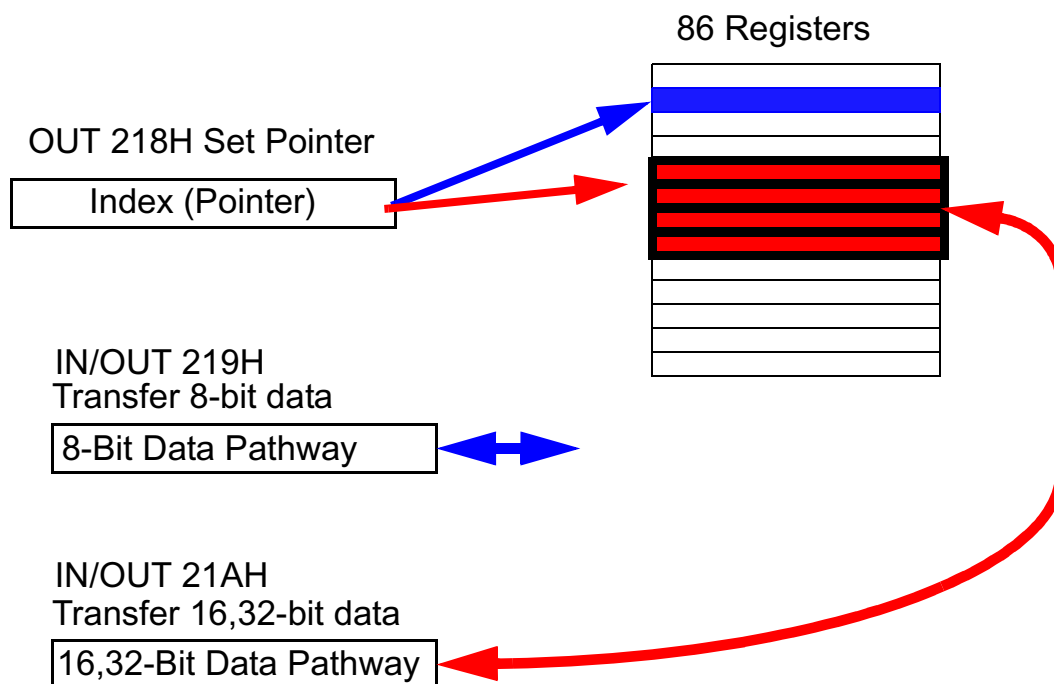
a] \_\_\_\_\_  
\_\_\_\_\_

3] PROGRAMMERS: Write Code to Read Revision into CX register (asm) or 16-bit unsigned int (C)

a] \_\_\_\_\_  
\_\_\_\_\_

[ ] Check Your Answers on [page 117](#).

# 87 ZF-Logic Register Space Access (16, 32-Bit)



Index	8-Bit Data at Index	8-Bit Data at Index + 1	
02	ZF-Logic Revision (LSB) -- (02H)	ZF- Logic Revision (MSB) -- (03H)	

- ZF Logic is Controlled or Accessed through about 86 8-bit registers.
- Many of these registers are logically 16 or 32-bits wide
- By setting 218H to put to a 16 or 32-bit base, 2 or 4 consecutive bytes from the ZF register space can be transferred with a single IN or OUT (or C inp or outp) instruction.

**Example:** Read the 16-Bit Data at Index 02 to pick up the ZF-Logic Revision:

```

mov  al,02h    ; Index
mov  dx,218h   ; Index Address
out  dx,al     ; Set Index
                ; read the value
mov  dx,21Ah   ; Data Viewport
in   ax,dx     ; AX=1234H (current revision of ZF-Logic)
    
```

# 88 ZF-Logic Registers

Index	8-Bit Data at Index	8-Bit Data at Index + 1	
02	ZF-Logic Revision (LSB) -- (02H)	ZF- Logic Revision (MSB) -- (03H)	
04	PWM Prescaler Low Byte -- (04H)	PWM Prescaler High Byte - (05H)	↑ PWM ↓
06	PWM duty cycle -- (06H)		
08	PWM I/O Control -- (08H)		
0A	PWM Read Output -- (0AH)		
0C	Watchdog 1 Count Low Byte -- (0CH)	Watchdog 1 Count High Byte -- (0DH)	↑ W/D ↓
0E	Watchdog 2 Count Value -- (0EH)	Watchdog Reset Pulse Length -- (0FH)	
10	Watchdog Control Low -- (10H)	Watchdog Control High -- (11H)	
12	Watchdog Events -- (12H)		
14	I/O Window 0 Base Low (14H)	I/O Window 0 Base High (15H)	↑ I/O Window ↓
16	I/O Window 0 Control (16H)		
18	I/O Window 1 Base Low (18H)	I/O Window 1 Base High (19H)	
1A	I/O Window 1 Control (1AH)		
1C	I/O Window 2 Base Low (1CH)	I/O Window 2 Base High (1DH)	
1E	I/O Window 2 Control (1EH)		
20	I/O Window 3 Base Low (20H)	I/O Window 3 Base High (21H)	
22	I/O Window 3 Control (22EH)		
24			

# 89 ZF-Logic Registers (2/3)

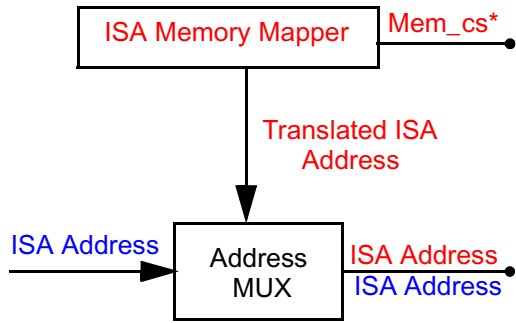
26	Memory Window 0 Base Bits 7-0	MW0 Base 15-12	MW0 Base 11-8
28	Memory Window 0 Base Bits 23-16	Memory Window 0 Base Bits 31-24	
2A	Memory Window 0 Size Bits 7-0	MW0 Size 15-12	MW0 Size 11-8
2C	Memory Window 0 Size Bits 23-16	Memory Window 0 Size Bits 31-24	
2E	Memory Window 0 Page Bits 7-0	MW0 Page 15-12	MW0 Page 11-8
30	Memory Window 0 Page Bits 23-16	Memory Window 0 Page Bits 31-24	
32	Memory Window 1 Base Bits 7-0	MW1 Base 15-12	MW1 Base 11-8
34	Memory Window 1 Base Bits 23-16	Memory Window 1 Base Bits 31-24	
36	Memory Window 1 Size Bits 7-0	MW1 Size 15-12	MW1 Size 11-8
38	Memory Window 1 Size Bits 23-16	Memory Window 1 Size Bits 31-24	
3A	Memory Window 1 Page Bits 7-0	MW1 Page 15-12	MW1 Page 11-8
3C	Memory Window 1 Page Bits 23-16	Memory Window 1 Page Bits 31-24	
3E	Memory Window 2 Base Bits 7-0	MW2 Base 15-12	MW2 Base 11-8
40	Memory Window 2 Base Bits 23-16	Memory Window 2 Base Bits 31-24	
42	Memory Window 2 Size Bits 7-0	MW2 Size 15-12	MW2 Size 11-8
44	Memory Window 2 Size Bits 23-16	Memory Window 2 Size Bits 31-24	
46	Memory Window 2 Page Bits 7-0	MW2 Page 15-12	MW2 Page 11-8
48	Memory Window 2 Page Bits 23-16	Memory Window 2 Page Bits 31-24	
4A	Memory Window 3 Base Bits 7-0	MW3 Base 15-12	MW3 Base 11-8
4C	Memory Window 3 Base Bits 23-16	Memory Window 3 Base Bits 31-24	
4E	Memory Window 3 Size Bits 7-0	MW3 Size 15-12	MW3 Size 11-8
50	Memory Window 3 Size Bits 23-16	Memory Window 3 Size Bits 31-24	
52	Memory Window 3 Page Bits 7-0	MW3 Page 15-12	MW3 Page 11-8
54	Memory Window 3 Page Bits 23-16	Memory Window 3 Page Bits 31-24	

↑  
Memory Window  
↓

# 90 ZF-Logic Registers (3/3)

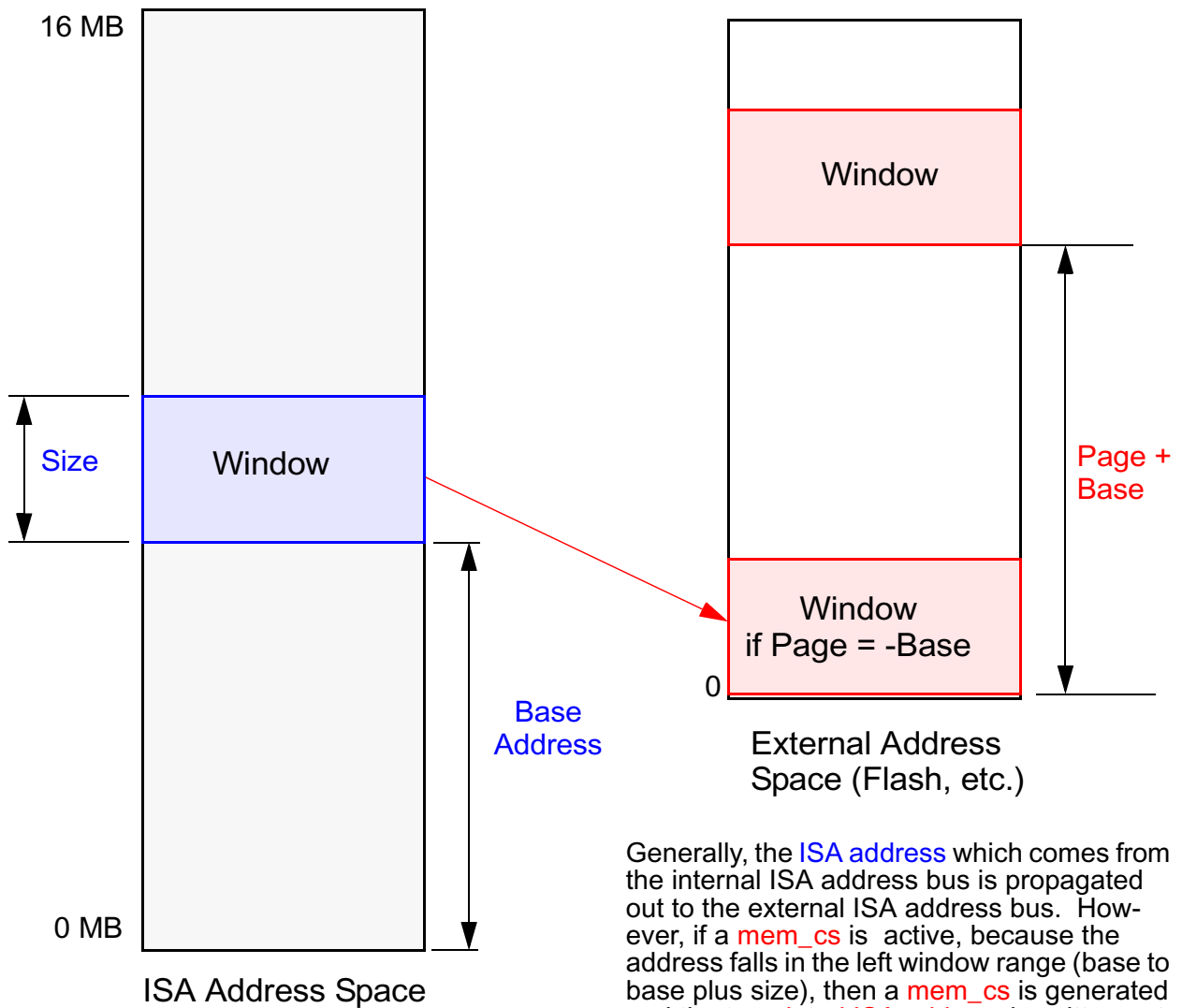
56		BUR Base Low (57H)	
58	BUR Base High (58H)		
5A	Memory Control Low (5AH)	Memory Control High (5BH)	
5C			
5E	Z-tag Data Write Register (5EH)		
60	Z-tag Data Read Register (60H)		
62	Bootstrap Bits 7-0 (62H)	Bootstrap Bits 15-8 (63H)	
64	Bootstrap Bits 23-16 (64H)		
66	I/O+Memory Window Map Events 66H		
68	Scratch Register 0 Low (68H)	Scratch Register 0 High (69H)	Scratch Registers ↑ ↓
6A	Scratch Register 1 Low (6AH)	Scratch Register 1 High (6BH)	
6C	Scratch Register 2 Low (6CH)	Scratch Register 2 High (6CH)	
6E	Scratch Register 3 Low (6EH)	Scratch Register 3 High (6FH)	
70	Scratch Register 4 Low (70H)	Scratch Register 4 High (70H)	
72	Scratch Register 5 Low (72H)	Scratch Register 5 High (73H)	
74	Scratch Register 6 Low (74H)	Scratch Register 6 High (75H)	
76	Scratch Register 7 Low (76H)	Scratch Register 7 High (77H)	
78	Scratch Register 8 Low (78H)	Scratch Register 8 High (79H)	
7A	Scratch Register 9 Low (7AH)	Scratch Register 9 High (7BH)	
7C	Z-tag control register (7CH)	Z-tag Sequencer Divisor Register (7DH)	
7E	Z-tag Sequencer Waveform (7EH)	Z-tag Sequencer Strobe Points (7FH)	
80	Z-tag Sequencer Data (80H)	Z-tag Sequencer Status (81H)	

# 91 ISA Memory Windows for Flash / SRAM



Memory Mapper Pins

PKG	Name	Description
B04	Mem_cs0	ZF-Logic Memory Mapper CS 0
D05	Mem_cs1	ZF-Logic Memory Mapper CS 1
A03	Mem_cs2	ZF-Logic Memory Mapper CS



Generally, the **ISA address** which comes from the internal ISA address bus is propagated out to the external ISA address bus. However, if a **mem\_cs** is active, because the address falls in the left window range (base to base plus size), then a **mem\_cs** is generated and the **translated ISA address** is written out to the physical address bus.



# 92 Benefits of Memory Window Mapping

- The ZFL allows the ZFx86 to control up to four external memory devices on the ISA bus. These devices can be mapped into the system memory address space. **Typically, this feature is used to map external Flash memory into the address space without external address decoding logic.**
- Each device can occupy up to 16 Megabytes (occupying all 24 ISA address lines).
- In DOS mode these windows can be up to 256K bytes and reside only in upper 1 Mbyte DOS ROM area (C0000-FFFFF).
- In protected mode the windows can occupy all 24 ISA address lines (000000 - FFFFFFFF). This area is accessed in protected mode through memory space above system SDRAM. If the address is not in the system memory and no PCI device claims it then it is forwarded to the ISA bus. This makes the ISA bus useful multiple times in the upper memory area.
- Memory can be **8 or 16 bits wide**, and **may be write protected**.
- note: This is Static RAM or Flash -- this is *not* the SDRAM Controller memory.

**Memory Control Low -- Index 5AH**

Bit	7	6	5	4	3	2	1	0
Function	w3_ro	w2_ro	w1_ro	w0_ro	w3_8	w2_8	w1_8	w0_8
Default	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Name	Function
7:4	wn_ro	Window n Read-write Control 0: Window N Is Read-write 1: Window N Is Read-only
3:0	wn_8	Window n Data Bus Width 0: Window N Uses 16-bit Data Access 1: Window N Uses 8-bit Data Access

# 93 Safety Aspects of Memory Windows

- Programmer/OS can request [interrupts on Memory Window Overlap, Memory Change](#).
- Memory can be 8 or 16 bits wide, and [may be write protected](#).
- Design Idea: Part of a Flash can be Write Protected by using two mem\_cs\* signals - one set R/W and one set R/O.

note: in case of overlapping addresses, mem\_cs will *not* be asserted. The effect of the Events register is to cause notification via interrupt of this problem.

**I/O and Memory Window Mapper Events -- Index 66H**

Bit	7	6	5	4	3	2	1	0
Function	Reserved		Event Type		Memory Overlap	I/O Overlap	Memory Window Change	I/O Window Change
Default	0		0		0	0	0	0
R/W	R/O		R/W		R/W	R/W	R/W	R/W

Bit	Name	Function
7:6	Reserved	
5:4	Event Type	Generated event type 00 - No event 01- SCI 10 - NMI 11 - SMI
3	Memory Overlap	Enable resolve event on memory overlap 0: Disable event on memory overlap 1: Enable event on memory overlap
2	I/O Overlap	Enable event on I/O window overlap 0: Disable event on I/O window overlap 1: Enable event on I/O window overlap
1	Memory Access	Enable event on memory window change 0: disable event 1: enable event
0	I/O Access	Enable event on I/O window change 0: disable event 1: enable event

# 94 Memory Window Registers

## 1. First window settings

- 2CH-2BH: Mem\_cs0 window size      0 - FFF000 = 16 MB / **0FFFFH**
- 30H-2FH: Mem\_cs0 page              0 - FFF000 = 16 MB / **0**
- 28H-27H: Mem\_cs0 base address    0 - FFF000 = 16 MB / **F0000H**

## 2. Second window settings

- 38H-37H: Mem\_cs1 window size
- 3CH-3BH: Mem\_cs1 page
- 34H-33H: Mem\_cs1 base address

## 3. Third window settings

- 44H-43H: Mem\_cs2 window size
- 48H-47H: Mem\_cs2 page
- 40H-3FH: Mem\_cs2 base address

## 4. Fourth window settings

- 50H-4FH: Mem\_cs3 window size
- 54H-53H: Mem\_cs3 page
- 4CH-4BH: Mem\_cs3 base address

## 5. 5BH-5AH: Control (R/W, 8/16 Width)

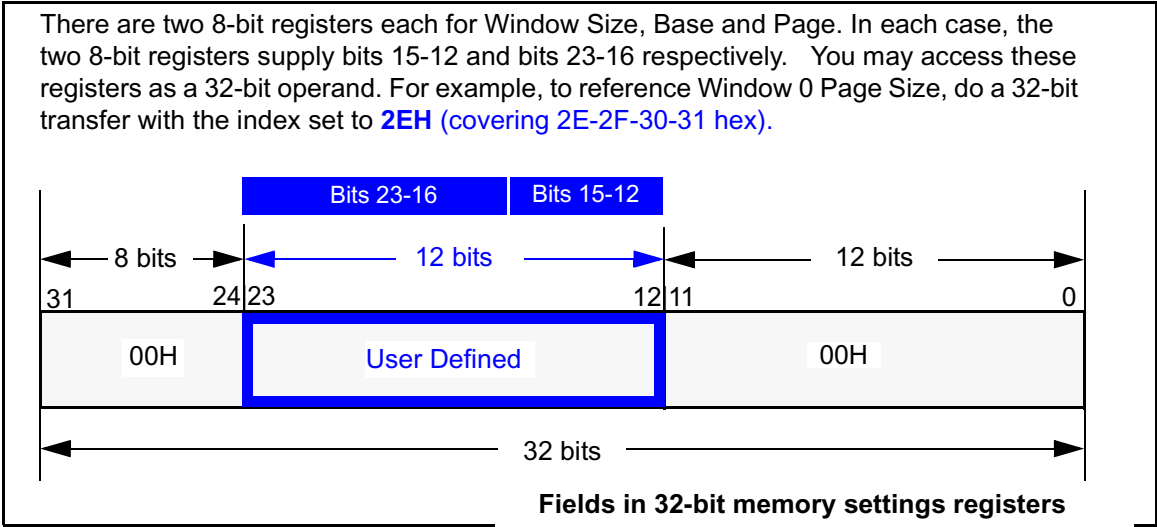
## 6. 66H: Events (SMI, etc.)

**Initialization  
Special Use of Mem\_cs0**

All the windows settings are initialized to 0 on power up reset. Setting a window size to 0 disables the window (the mem\_cs\*).

**mem\_cs0** is generally used for the boot ROM (flash). Thus the first window reset defaults are as shown above.

Window Size, Base Address, Page (translated address) all have a range of 0-16 MB. The mapping is disabled when *window size* is zero.



notes: SMI (System Management Interrupt) is used for legacy SMM BIOS; SCI (System Control Interrupt) is used for ACPI OS. Also, when determining whether or not an addresses emitted by the processor should generate a mem\_cs\*, the upper 8-bits of the 32-bit memory address are ignored. Thus a memory page can appear (alias) many times in the (CPU's) 4 GB address space.

# 95 ZF-Logic Memory Windows Review Questions

---

4] Looking at the ['ZF-Logic Block Diagram' on page 85](#), what is the difference between the internal ISA bus and the output (on the lower right) called "ISA Address"? What is the function of the address multiplexer shown in the diagram, and when is the operative?

a] \_\_\_\_\_  
\_\_\_\_\_

5] What are the necessary and sufficient conditions to cause a mem\_CS signal to be asserted by the ZFx86? Even if these conditions are met, when will a mem\_cs signal not be asserted?

a] \_\_\_\_\_  
\_\_\_\_\_

6] How many 32-bit ZF-Logic "memory window" registers are associated with the four mem\_cs signals? See ['ZF-Logic Registers \(3/3\)' on page 90](#).

a] \_\_\_\_\_  
\_\_\_\_\_

7] List the benefits to the customer which accrue from the ZFx86 memory window mapping feature.

a] \_\_\_\_\_  
\_\_\_\_\_

8] What is the benefit to the customer due to the fact that his operating system may be notified based on memory window change? How this notification occur? See "I/O and memory window mapper events -- index 66 hex" on page 14.

a] \_\_\_\_\_  
\_\_\_\_\_

9] What a special about mem\_cs0? In order for that to work, what must happen?

a] \_\_\_\_\_  
\_\_\_\_\_

[ ] Check Your Answers on Page [page 118](#).



**GPCS Pins**

PKG	Name	Description
B03	io_cs0	ZF-Logic I/O Mapper GPCS 0
A02	io_cs1	ZF-Logic I/O Mapper GPCS 1
A01	io_cs2	ZF-Logic I/O Mapper GPCS 2
C03	io_cs3	ZF-Logic I/O Mapper GPCS 3

- ZFx86 has four GPCS (General Purpose Chip Select) signals mapped to io\_cs\* pins.
- Each io\_cs\* signal is assigned an address (or a set of consecutive addresses) in the ISA I/O space. This address, or set of consecutive addresses, is called the "window".
- Benefits: Each I/O Chip, requiring 1 to 16 consecutive I/O addresses, can be connected to the ZFx86 without glue logic.
- You can specify read-only or read-write, and 8 or 16-bit wide data transfers.
- Benefit: if your I/O chip is 8 or 16-bits wide, you do not need special logic to inform the ZFx86 -- it is set up in the control registers.
- There is no address translation here: just appropriate io\_cs\* generation.

notes: For example, if the chip you wish to connect to the ZFx86 using one of the io\_cs pins has four ports (such as the old 8255 chip), you would want the chip select to be asserted for four consecutive addresses. The chip itself would differentiate between the addresses by looking at the low two bits of the ISA address bus

# 97 | GPCS (I/O Mapper) Register Set

---

## 1. GPCS 0 settings

- 15H-14H: io\_cs0 base address      0000 - FFFFH (CPU I/O Range)
- 16H:        io\_cs0 control            RO/RW, 8/16 bit data, size, enable

## 2. GPCS 1 settings

- 19H-18H: io\_cs0 base address
- 1AH:        io\_cs0 control

## 3. GPCS 2 settings

- 1DH-1CH: io\_cs0 base address
- 1EH:        io\_cs0 control

## 4. GPCS 3 settings

- 21H-20H: io\_cs0 base address
- 22H:        io\_cs0 control

## 5. Events Register (66H) Window Change, Overlap

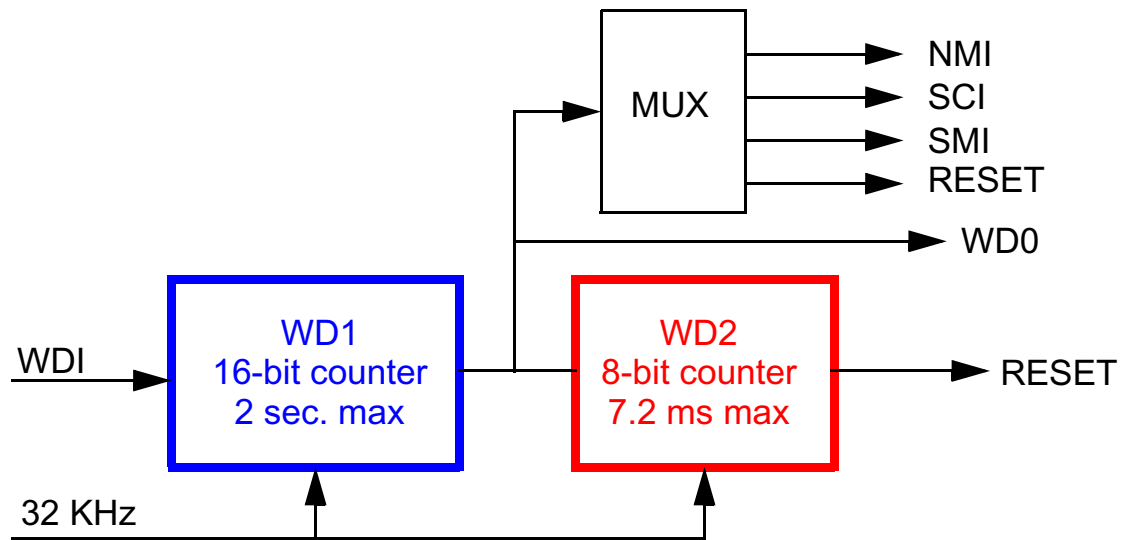
- A chip select may be generated for any 1 to 16 consecutive addresses in any part of the CPUs to 64K I/O address space. The Intel architecture supports only 64K of I/O address space, so the 16-bit base address provides precise and complete locating of the chip select decode.
- Sufficient control information is required on a per chip-select basis, to require one control byte for each I/O chip select.
- The same events register (66H) is used for the memory and I/O mapping. The same errors are detected.

# 98 GPCS (I/O Mapper) Control Registers

## I/O Window "N" Control

Bit	7	6	5	4	3	2	1	0
Function	win_ro	16_bit	act_lvl	win_en	win_siz: I/O Window Size (1-16)			
Default	0	0	0	0	0 (Size is 1)			
R/W	R/W	R/W	R/W	R/W	R/W			

Bit	Name	Function
7	win_ro	I/O window read/write control 0: Access is read-write 1: Access is read-only  Setting window to read-only mode disables IOW_N signal on ISA bus for IO window address range.
6	16_bit	I/O window datapath width 0: 8-bit wide access 1: 16-bit wide access
5	act_lvl	io_cs active level 0: io_cs is active low 1: io_cs is active high
4	win_en	I/O window enable in I/O space 0: I/O window is disabled 1: I/O window is enabled
3:0	win_siz	Number of consecutive 8-bit I/O addresses to decode starting from I/O window base.  The number of consecutive addresses decoded is win_siz + 1. For example, setting the window size to 0 enables one I/O address at I/O window base. Setting size to 0Fh will enable I/O window of 16 addresses starting from I/O window base.



- Whenever WD1 is not reloaded during a pre programmed interval it generates an event to notify the system of an error condition.
- The first watchdog timer is initialized to a 16-bit timeout value through registers 0Ch and 0Dh. After enabling through control register (10h) it starts the countdown to zero. The first watchdog timer can be reloaded to an initial value by writing into control register (10h) or asserting watchdog external control pin on ZFx86 (WDI).
- Whenever the first watchdog is not reloaded during the timeout value it generates an event to notify the system of an error condition and outputs the logical "1" to a watchdog output pin on ZFx86 (WDO). The notification event can be routed to NMI, SMI, SCI or it can reset the system immediately.
- The second watchdog timer 8-bit timeout value is initialized through register 0Eh and starts counting down after WD1 time-out. When the WD2 counter reaches zero, it will unconditionally cause system reset.



# 100 ZF-Logic Registers for the Watchdog Timer

0C	Watchdog 1 Count Low Byte (0CH)	Watchdog 1 Count High Byte (0DH)
0E	Watchdog 2 Count Value (0EH)	Watchdog Reset Pulse Length (0FH)
10	Watchdog Control Low (10H)	Watchdog Control High (11H)
12	Watchdog Status (12H)	

- **Count Registers** - Reload Values for Watchdog Timers
- **Watchdog Reset Pulse Length** - the number of 32kHz ticks to hold the system reset signal low
- Watchdog Control: Enable/Disable, and MUX Control

# 101 ZF-Logic Registers for the Watchdog Timer

Watchdog Control Low -- Index 10H

Bit	7	6	5	4	3	2	1	0
Function	reserved		wd2 load	wd1 load	reserved		wd2 enable	wd1 enable
Default	0		0	0	0		0	0
R/W	R/O		R/W	R/W	R/O		R/W	R/W

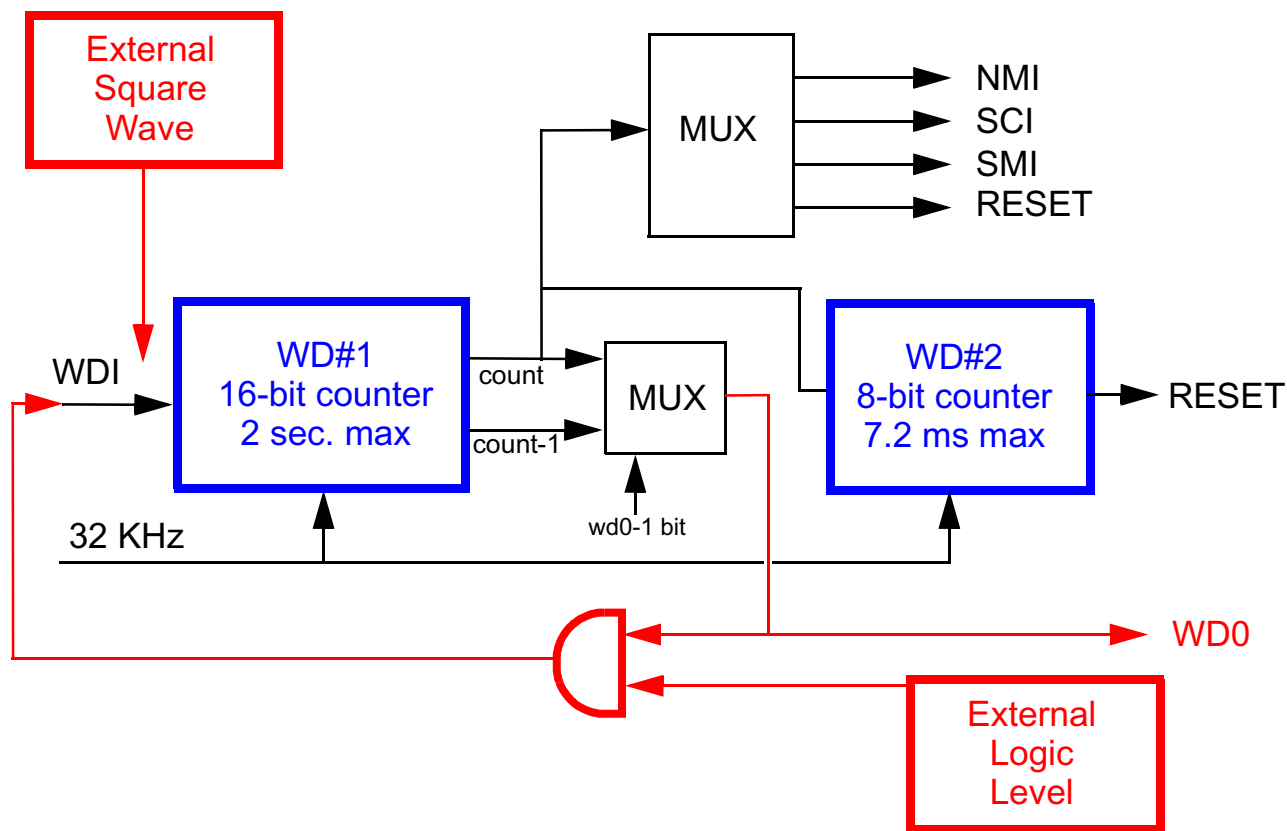
Bit	Name	Function
7:6	Reserved	
5	wd2 load	Reload WD2 counter. Active event for this bit is transition from 0 to 1
4	wd1 load	Reload WD1 counter. Active event for this bit is transition from 0 to 1
3:2	Reserved	
1	wd2 enable	Enable wd2 0: WD2 is disabled 1: WD2 is enabled
0	wd1 enable	Enable wd1 0: WD1 is disabled 1: WD1 is enabled

# 102 ZF-Logic Registers for the Watchdog Timer

Watchdog Control High -- Index 11H

Bit	7	6	5	4	3	2	1	0
Function	reserved	wdi_en	wdo_-1	wdi edge	wd1 reset	wd1 SMI	wd1 NMI	wd1 SCI
Default	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Name	Function
7	Reserved	
6	wdi_en	<b>Enable the assertion of WDI input pin on ZFx86 to to reload the watchdog 1 counter</b> 0: WDI input ignored 1: WDI assertion reloads watchdog 1 counter
5	wdo_-1	Create output on WDO output pin on ZFx86 at WD1 time-out or one 32kHz clock tick before 0: WDO signal will be set high on WD1 expiration 1: WDO signal is set high one clock tick before WD1 expires. WD1 events will always occur at WD1 time-out and are not affected by wdo_-1 bit setting.  This feature permits <b>automatic reload</b> of WD1 when WDO is wired to WDI.
4	wdi edge	Active front of WDI input 0: WDI is asserted on 0->1 transition 1: WDI is asserted on 1->0 transition
3	wd1 reset	WD1 generates <b>system reset</b> on time-out 0: WD1 will not generate system reset on time-out 1: WD1 will generate system reset on time-out
2	wd1 SMI	WD1 generates <b>SMI</b> on time-out 0: wd1 will not generate SMI on time-out. 1: wd1 generates SMI on time-out
1	wd1 NMI	WD1 generates <b>NMI</b> on time-out 0: wd1 will not generate NMI on time-out 1: wd1 generates NMI on time-out
0	wd1 SCI	WD1 generates <b>SCI</b> on time-out 0: wd1 will not generate SCI on time-out 1: wd1 generates SCI on time-out



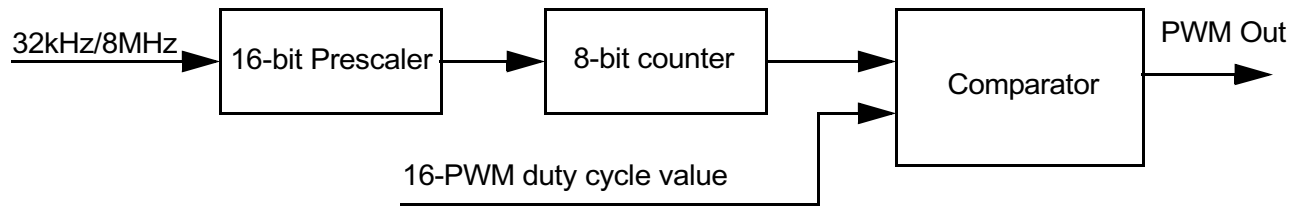
- **Square Wave** on WDI may reset (external reset)
- Wiring WDO to WDI through Gate allows use of **Logic Level** rather than Square Wave
- When wiring WDO to WDI, to prevent event (NMI, etc) (so long as gate is on) then use wdo-1 to set reset of WD#1 1 event before expiration

Notes: There is a WDO and WDI pin -- the WDI can be programmed to reload WD#1. If you toggle WDI (falling or rising) you can prevent the WD from ever expiring. The benefit of this is that so long as an external square wave is coming in, the WD never expires. You are thus using an EXTERNAL way of keeping the ZFx86 from resetting -- you are watching for an external "dead man" switch.

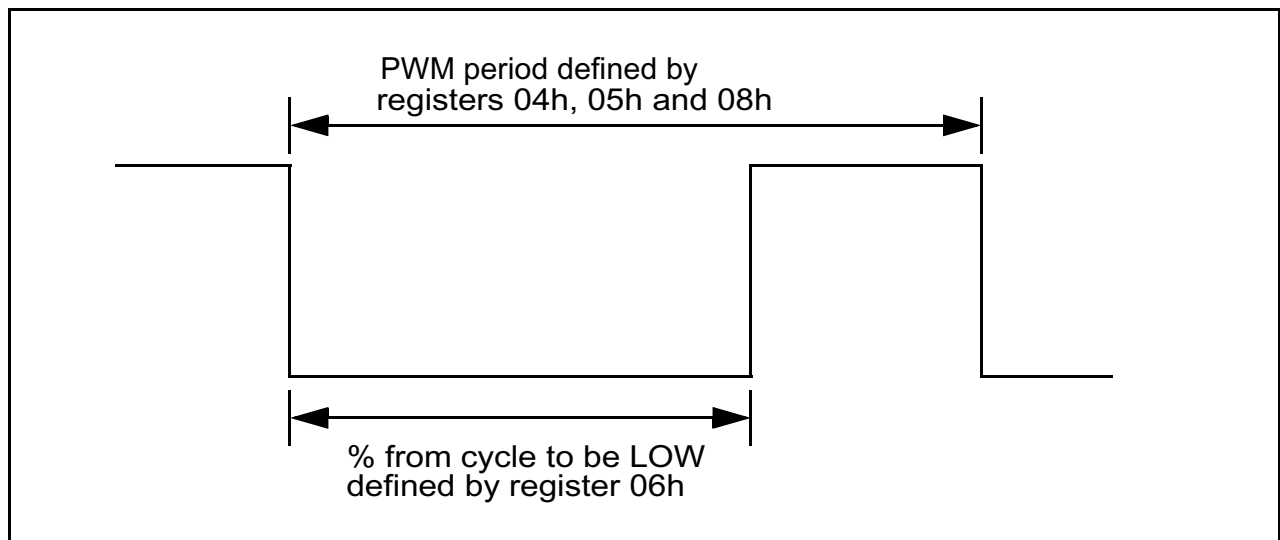
To do this, you need to have an outside event generator. Let's assume that all you have is a logic signal which shows you if the external system is working or not. You can then connect WDO to WDI with an OR gate or AND gate to that external signal.

If you set it up this way, then you set wdo-1 to generate event 1 pulse before expiring. If you did not do this, it would expire. See wdo-1 in ["Watchdog Control High -- Index 11H" on page 102](#)

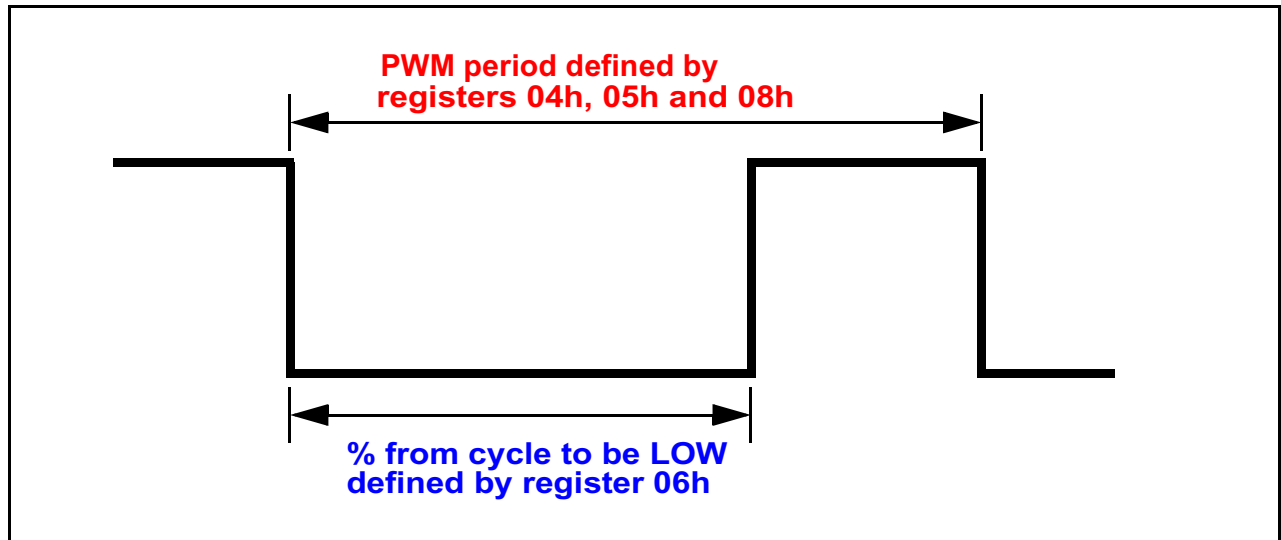
# 104 PWM Generator



- The PWM (Pulse Width Modulation) output may be used to create DC control voltage for an LCD backlight or any other device that requires this feature. The conversion is done by integrating variable duty cycle signal externally. At higher frequencies it may be used to control external transformer for DC/DC conversion.



# 105 PWM Generator Period / Duty Cycle



ZF-Logic Index for the PWM Generator

04	PWM Prescaler Low Byte -- (04H)	PWM Prescaler High Byte - (05H)
06	PWM duty cycle -- (06H)	
08	PWM I/O Control -- (08H)	
0A	PWM Read Output -- (0AH)	

- **PWM Prescaler:** Divides 8MHz or 32kHz input clock selected at PWM control register. Actual divisor is 16-bit PWM divisor word (combined of registers 04h and 05h) + 1
- **PWM Duty Cycle:** Sets the % of the cycle to be low. (0 = 100%, 255 = 0%).
- **PWM I/O Control:** Includes selection of 32kHz clock or 8 MHz ISA clock

# 106 PWM Generator - I/O Control Register

**PWM I/O Control -- Index 08H**

Bit	7	6	5	4	3	2	1	0
Function	reserved		enable direct	direct output	reserved		slow-fast clksrc	Enable PWM
Default	0		0	0	0		0	0
R/W	R/O		R/W	R/W	R/O		R/W	R/W

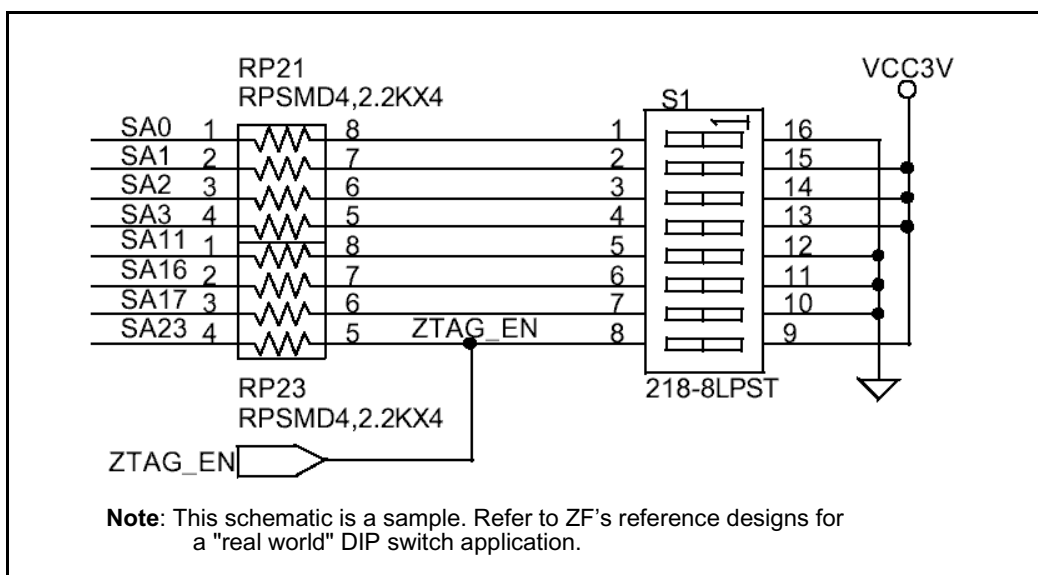
Bit	Name	Function
7:6	Reserved	
5	enable direct	Enables direct control of PWM output by bit 4 0: PWM drives the output 1: Bit 4 of register 08H drives the PWM output pin
4	direct output	The value of PWM output when bit of register 08h is set to 1
3:2	Reserved	
1	slow-fast (clksrc)	Selects the PWM prescaler input clock 1: PWM is clocked by 32kHz clock 0: PWM is clocked by 8 MHz ISA clock
0	Enable/Disable PWM	Enable/Disable PWM output 0: PWM is disabled 1: PWM is enabled

# 107 Boot Parameters Register

- When **power-on reset** is asserted 24 signals are read into the Boot Parameters Register (configuration register) from the ISA Address Bus.
- In a typical design, DIP switches or jumpers are used (with appropriate resistors) set to the bits in the Boot Strap Register.

ZF-Logic Index for the Boot Parameters Register

62	Bootstrap Bits 7-0 (62H)	Bootstrap Bits 15-8 (63H)
64	Bootstrap Bits 23-16 (64H)	



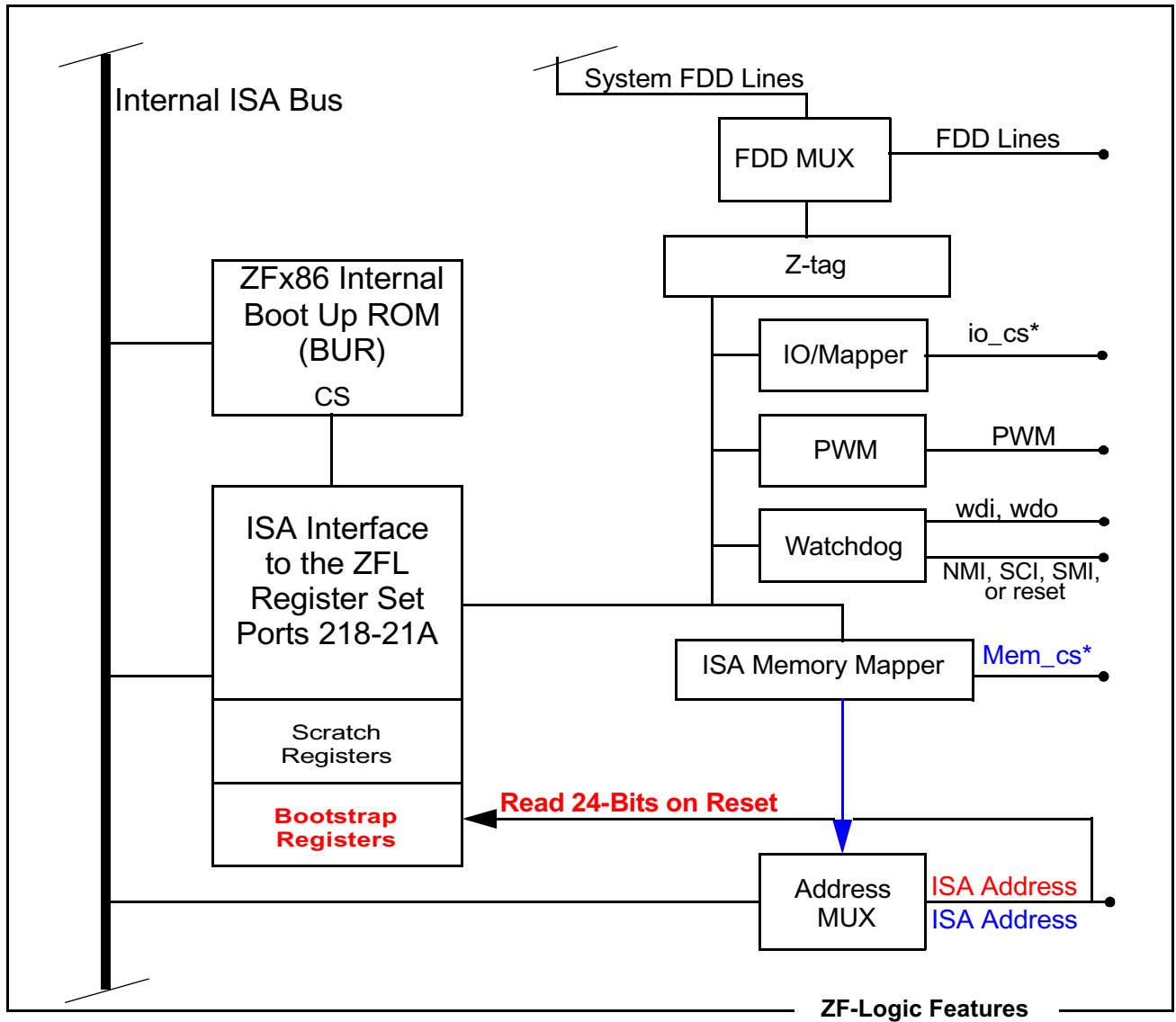
The ISA address bus (pins SA0-SA23) is tri-stated during the reset pulse. It contains on-chip weak (about 20K) pull-ups and pull-downs to set the default state of the bootstrap register. To override this, we use a 2.2K pull-up or pull down and a DIP switch or jumper. Once the reset pulse is done, the ISA bus has sufficient drive to overcome the effect of these 2.2K resistors.

Thus, conceptually the ISA address bus has three "modes": (1) the weak on-chip pull-ups/pull-downs which are operative during the tri-state; (2) the 2.2K pull-ups/pull-downs which may be activated via DIP switches; and (3) the normal execution time mode where the drive of the ISA address bus will override these resistors.

Since the Boot Parameters Register is read only, the values sampled on the ISA bus on the trailing edge of reset are "permanent" until the next hardware reset. Software can read the data which is latched, but cannot change the data in the bootstrap registers.



# 108 Boot Parameters Register (Continued)



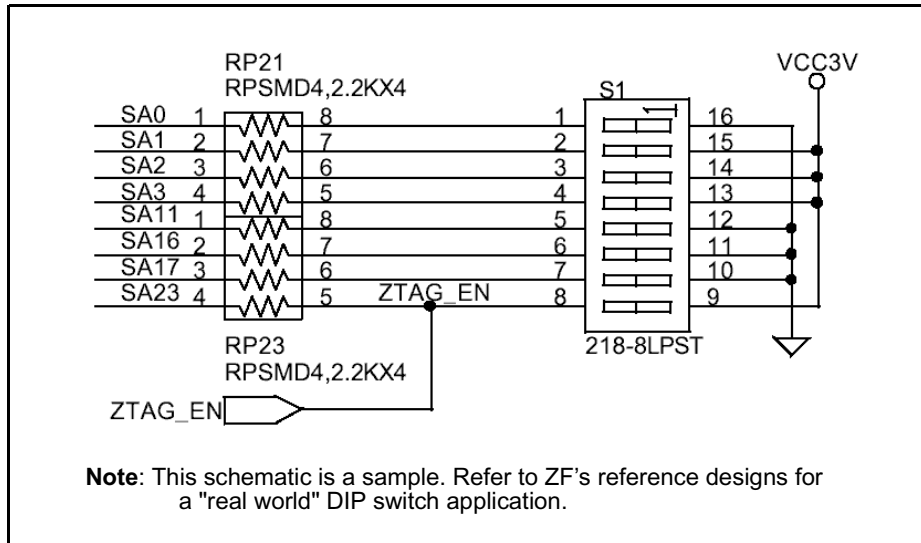
Example:

; read bootstrap registers as 32-bit value into EAX

```

0102 B0 62      mov  al,62h
0104 BA 0218   mov  dx,ZFLINDEX
0107 EE       out  dx,al
0108 BA 021A   in   eax,dx
010D 66| 25 00FFFFFF and  eax,0FFFFFFh
    
```

# 109 Boot Parameters Register (Continued)



## Sample DIP Switch Settings

SW	Function
1	ON - USER DEFINED
2	ON - USER DEFINED
3	ON - USER DEFINED
4	ON - USER DEFINED
5	ON - EXT ROM when Z-tag enabled <sup>a</sup>
6	System Clock Speed
7	System Clock Speed
8	ON - Boot from BUR

a. This bit should NOT be used in real designs. It is for testing only.

ISA BIT	Index	Bit	Name	Default	Function
23	64H	7	Boot from BUR (sometimes called ZTAG_EN)	0	Boot from BUR 1 = Boot from BUR 0 = Boot from Flash

When you plug the dongle in, it automatically sets BS23.

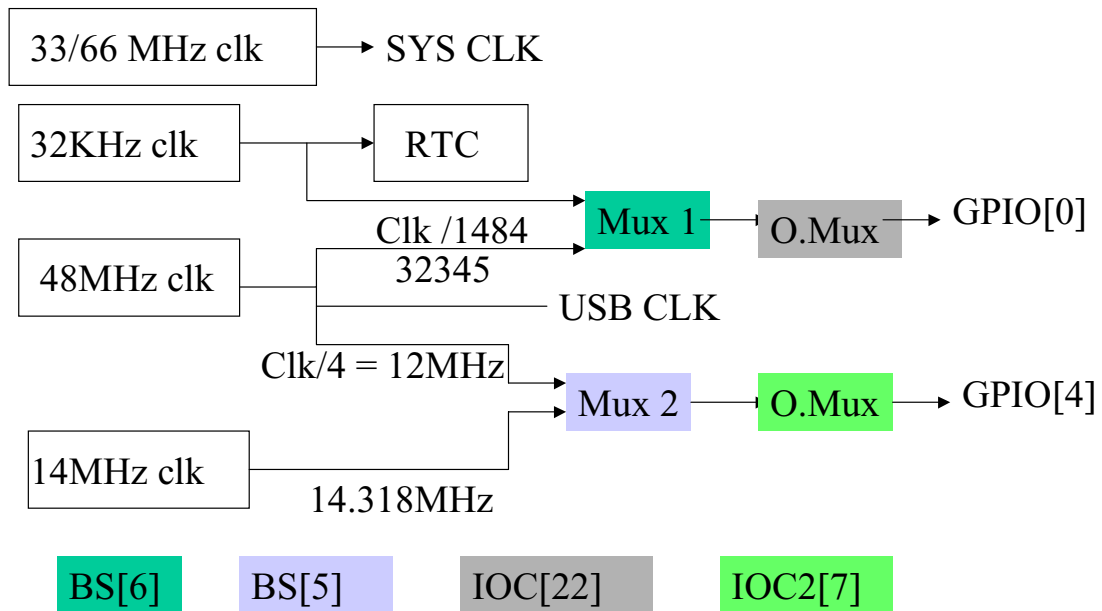
# 110 Boot Parameters Register (Continued)

Composite BootStrap Register Map

ISA BIT	Index	Bit	Name	Def	Function
0-3	62H	0-3	User Defined	0	User Defined
5	62H	5	14 Mhz clock source	0	14MHz Clock Source If 1, derive from 48Mhz. If 0, use mhz14_c pin. [AF16]
6	62H	6	32 KHz	0	32KHz Clock Source If 1, derive for 48MHz. If 0, use 32KHZC [AF01]
9	63H	1	3 <sup>rd</sup> PCI Request	0	Third PCI Request/Grant 1 = drq1 = req2_n and dack1_n = gnt2_n
11	63H	3	Reserved	1	Internal / External BUR Source. 0 = External BUR 1 = Internal BUR
12	63H	4	ISA Boot ROM Width	1	ISA Boot ROM Width 0 = 16 bit 1 = 8 bit
16 17	64H	0, 1	486 Clk Multiply	11	00 - Sys Clk * 1 01 - Sys Clk * 2 11 - Sys Clk * 3 (default) 10 - Sys Clk * 4
18	64H	2	FPCI divide	0	Frontside PCI Clock Divide. 0- SysClk 1 - SysClk / 2...
19	64H	3	BPCI divide	0	Backside PCI Clock Divide. <sup>a</sup> 0- SysClk 1 - SysClk / 2..
20	64H	4	BPCI Select	1	Backside PCI Clock Select. <sup>a</sup> 0 - External clock. 1- Internal clock.
23	64H	7	Z-tag enable	0	Causes BUR Boot. Enables the Z-Tag Interface and BUR if high.

a. If Bit 20 is 1, then Bit 19 has no affect.

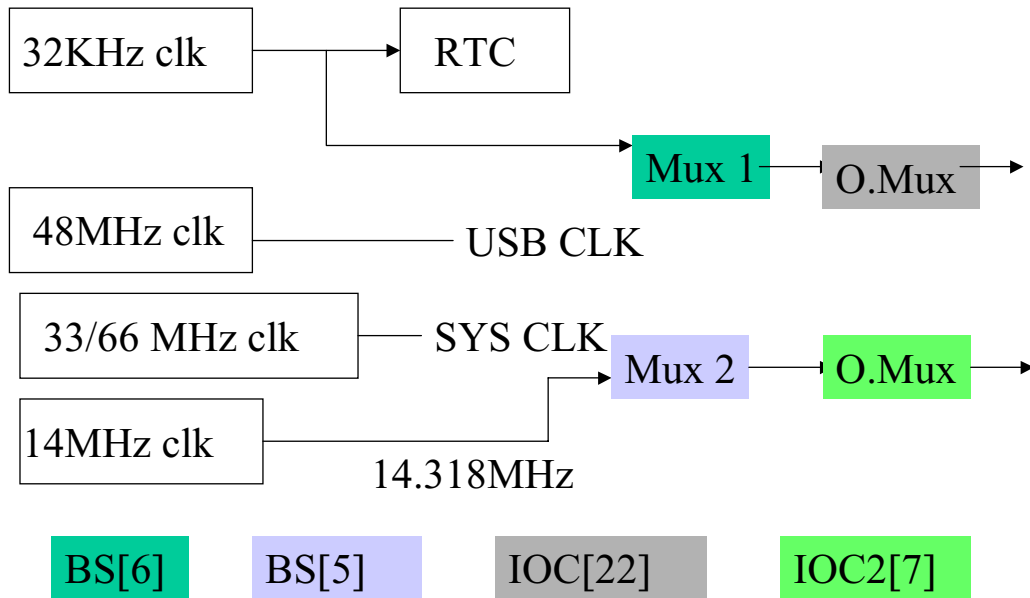
## Full Clocking Diagram



ISA BIT	Index	Bit	Name	Def	Function
5	62H	5	14 Mhz clock source	0	14MHz Clock Source If 1, derive from 48Mhz. If 0, use mhz14_c pin. [AF16]
6	62H	6	32 KHz	0	32KHz Clock Source If 1, derive for 48MHz. If 0, use 32KHZC [AF01]

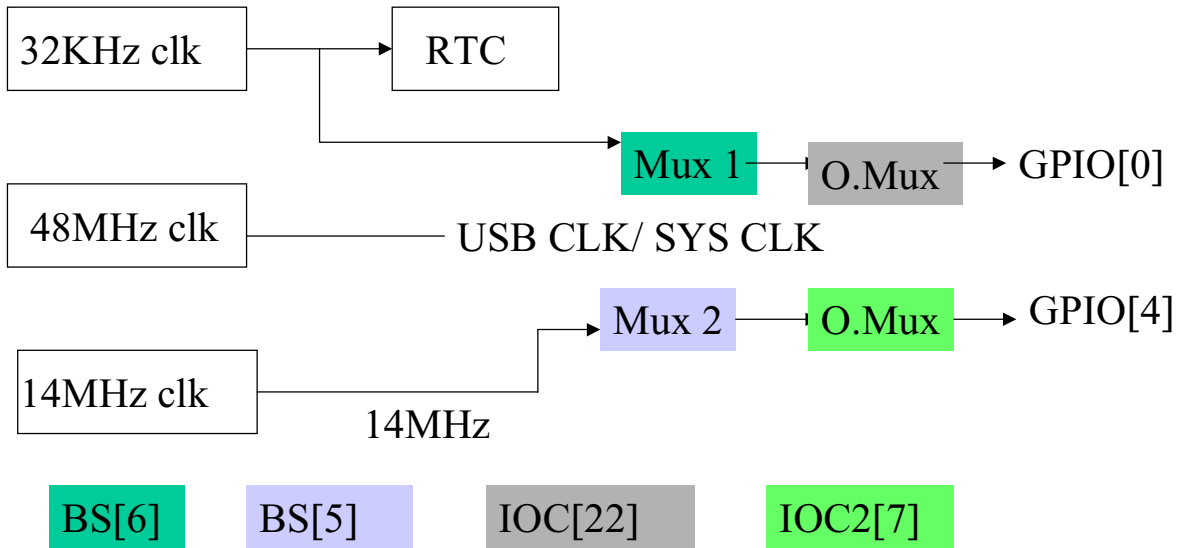
NOTES: The ZFx86 System-on-a-Chip has various clocking options. These options represent different trade-offs that the designer must investigate to come to the best solution for the application being considered. Essentially, the chip can be clocked using as many as four sources or as few as one.

# 112 Clocking Choices (1)



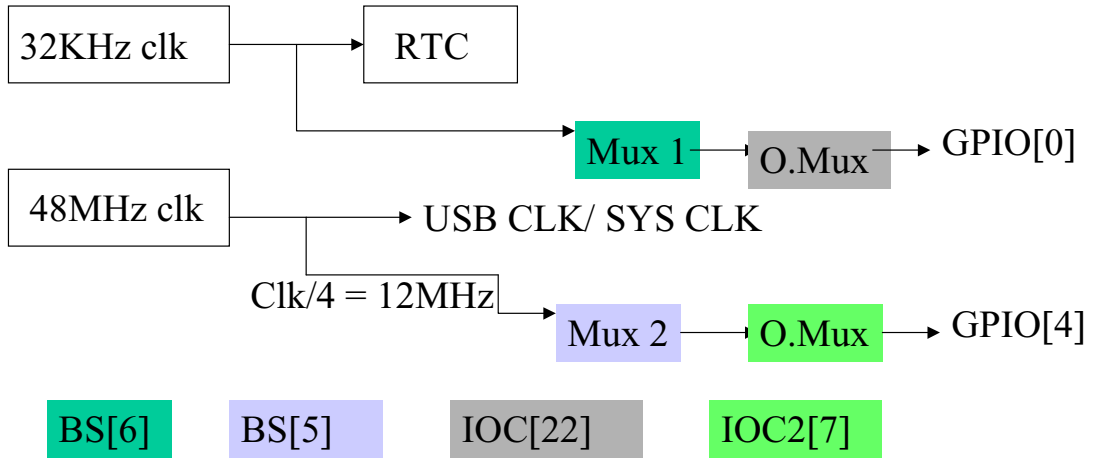
# 113 Clocking Choices (2)

**• LIMITED OPTIONS ON DRAM AND CPU CLOCK.**



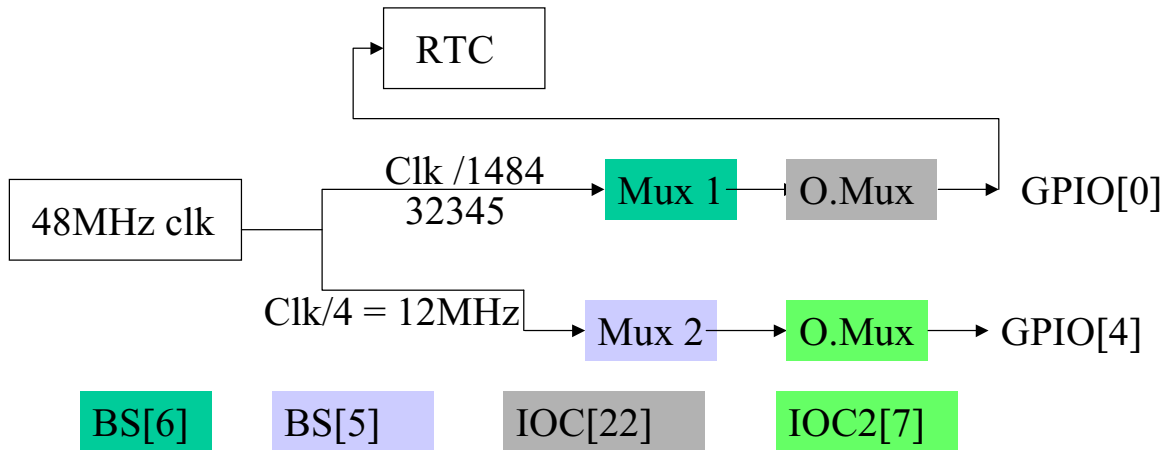
# 114 Clocking Choices (3)

- LIMITED OPTIONS ON DRAM AND CPU CLOCK, ISA CLOCKING OFF.



# 115 Clocking Choices (4)

- Simplest clock choice: Lose RTC on Power down, incur error in time, and ISA timing.





## Performance

- What is the best clocking for the customer?

<b>Application</b>	<b>Key Clock</b>
<b>Kiosc</b>	<b>SYSCLK @ 66, CPU 2X, PCI /2</b>
<b>Taxi</b>	<b>SYSCLK @ 48 CPU 2X, 32KHZ</b>
<b>POS</b>	<b>SYSCLK @ 48, CPU 2X, 32KHz</b>
<b>Agriculture</b>	<b>SYSCLK 48 ONLY</b>

# Answers to Questions

<[page 86](#)>

- 1] Why do we have 86+ Registers in the ZF Logic?
  - a] The actual address range of the ZF-Logic registers is index 2 through 81H, or about 128 byte addresses. Of these, approximately 86 off the byte addresses have read/write data fields. We need this many registers to contain all of the control and status bits for all of the functions built into the ZF-Logic.
- 2] Why do we use 3 (rather than 86+) ISA Addresses?
  - a] The industry standard architecture, a derivative of the first IBM personal computer, has many assigned and reserved addresses within the I/O address space. In order to have a minimum profile, we access all of the ZFL control logic registers using three and only three ISA I/O addresses.
- 3] PROGRAMMERS: Write Code to Read Revision into CX register (asm) or 16-bit unsigned int (C)
  - a]

```
mov    al,02h    ; Index
mov    dx,218h   ; Index Address
out    dx,al     ; Set Index
                ; read the value
mov    dx,21Ah  ; Data Viewport
in     ax,dx     ; AX=1234H

; this solution cheats and uses a 16-
bit transfer that we have not covered
yet.
```

```
uchar ucLow, ucHigh;
unsigned int uiRevision;
#define INDEX 0x218
#define TRANSFER 0x219
#define REVISION_LSB 2
#define REVISION_MSB 3

// solve using 8-bit transfers

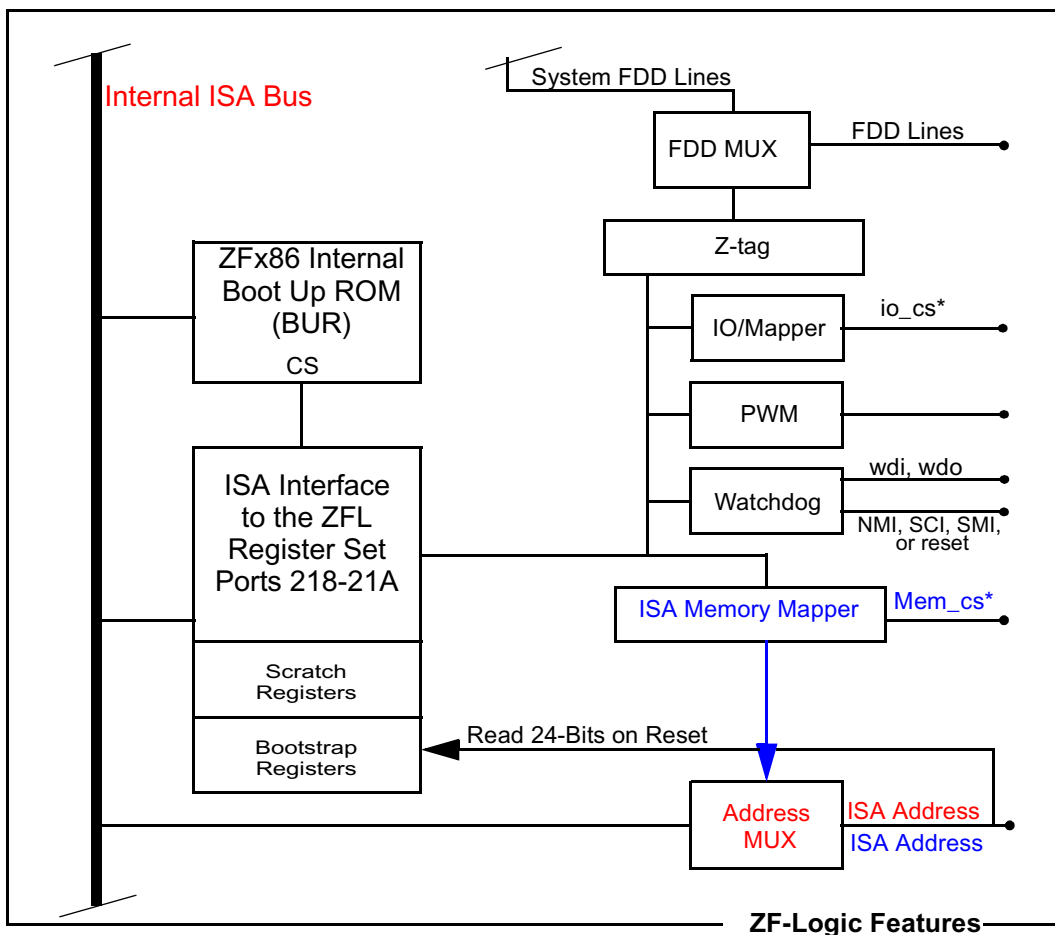
outp (INDEX, REVISION_LSB);
ucLow = inp (TRANSFER);
outp (INDEX, REVISION_MSB);
ucHigh = inp (TRANSFER);
uiRevision = ucHigh * 256 + ucLow;

// alternative using 16-bit transfer

#define TRANSFER1632 0x218

outp (INDEX, REVISION_LSB);
uiRevision = inpw (TRANSFER1632);
```

- 4] Looking at the ['ZF-Logic Block Diagram' on page 85](#), what is the difference between the internal ISA bus and the output (on the lower right) called "ISA Address"? What is the function of the address multiplexer shown in the diagram, and when is the operative?
- a] Addresses on the internal ISA bus are generally routed to the output pins on the ZF86 whenever there are memory read, memory write, input read, and output write, transfer cycles. That is, unless the [ISA memory mapper](#) grabs certain memory read and memory write cycles, the addresses from those cycles propagates directly out on the ISA address bus. However, if a memory address is in the range specified by the base to base + size, then two things happen: (1) the appropriate [mem\\_cs](#) is generated; and (2) the [ISA memory mapper](#) provides a translated address and that is the address which goes out on the external ISA address bus pins. Thus the function of the address multiplexer shown in the diagram is to route to the external ISA address bus either (1) [the ISA address from the internal ISA bus](#), or (2) the [translated ISA address from the ISA memory mapper](#).



5] What are the necessary and sufficient conditions to cause a mem\_CS signal to be asserted by the ZFx86? Even if these conditions are met, when will a mem\_cs signal not be asserted?

a] In order to get one of the four mem\_cs signals to be generated, a memory read or memory write generated by the CPU must (1) reach the internal ISA bus, and (2) be between the window base address and the window base+size.

Note that mem\_cs signal not be asserted in the window size equals 0 (as it turns off ISA memory mapping for that window), or if this memory window overlaps another memory window. Review ['ISA Memory Windows for Flash / SRAM' on page 91](#).

6] How many 32-bit ZF-Logic "memory window" registers are associated with the four mem\_cs signals? See ['ZF-Logic Registers \(3/3\)' on page 90](#).

a] There are 3 32-bit registers per memory window or a total of 12 32-bit registers. These registers contained the base, size, and target offset (page). There are other control registers (a control and an event register), but the basic window mapping occurs using the 3 32-bit registers per window.

7] List the benefits to the customer which accrue from the ZFx86 memory window mapping feature.

a] The first, and most important benefit, is that no extra hardware or glue logic is required in order to manage the chip selects for up to 4 SRAM or flash devices. In addition, each device may be programmed to be read-write or read-only. Further, without any extra external logic, a data bus width for each device may be specified to be 8 or 16 bits. Besides that, there is an event register which enables various interrupts to be received by the operating system based on misuse of the memory mapping feature.

8] What is the benefit to the customer due to the fact that his operating system may be notified based on memory window change? How this notification occur? See ['I/O and Memory Window Mapper Events -- Index 66H' on page 93](#).

a] If an unauthorized program purposely or accidentally attempts to change the size of a memory window it can represent a serious bug causing product release and development delays (during development) or subsequent failures in the field. Well behaved hardware will notify the the operating system via interrupts if something happens which is not supposed happen. The ability of the ZFx86 to notify the operating system via interrupts if someone tries to change the memory window parameters, or if there is an inadvertent overlap of the memory windows, enhances system integrity.

9] What is special about mem\_cs0? In order for that to work, what must happen?

a] When the CPU powers up, it does an instruction fetch from 000FFFF0H, which in all Intel x86 computers is routed to the last 16 bytes of the boot ROM. In ZFx86 designs, we connect mem\_cs0 to the flash chip containing the boot code. The ZFx86 chip, on power-up reset, initializes 3 32-bit registers for mem\_cs0 such that this instruction fetch will read from a flash device.

As a technical note: The window size is set to 64 K, and the base address to F0000 (the last 64K of the 1MB of "real" memory), and the page to 0. This means that instruction fetches in the top 64K of the 1MB real address space will read from the first 64K of the flash chip.

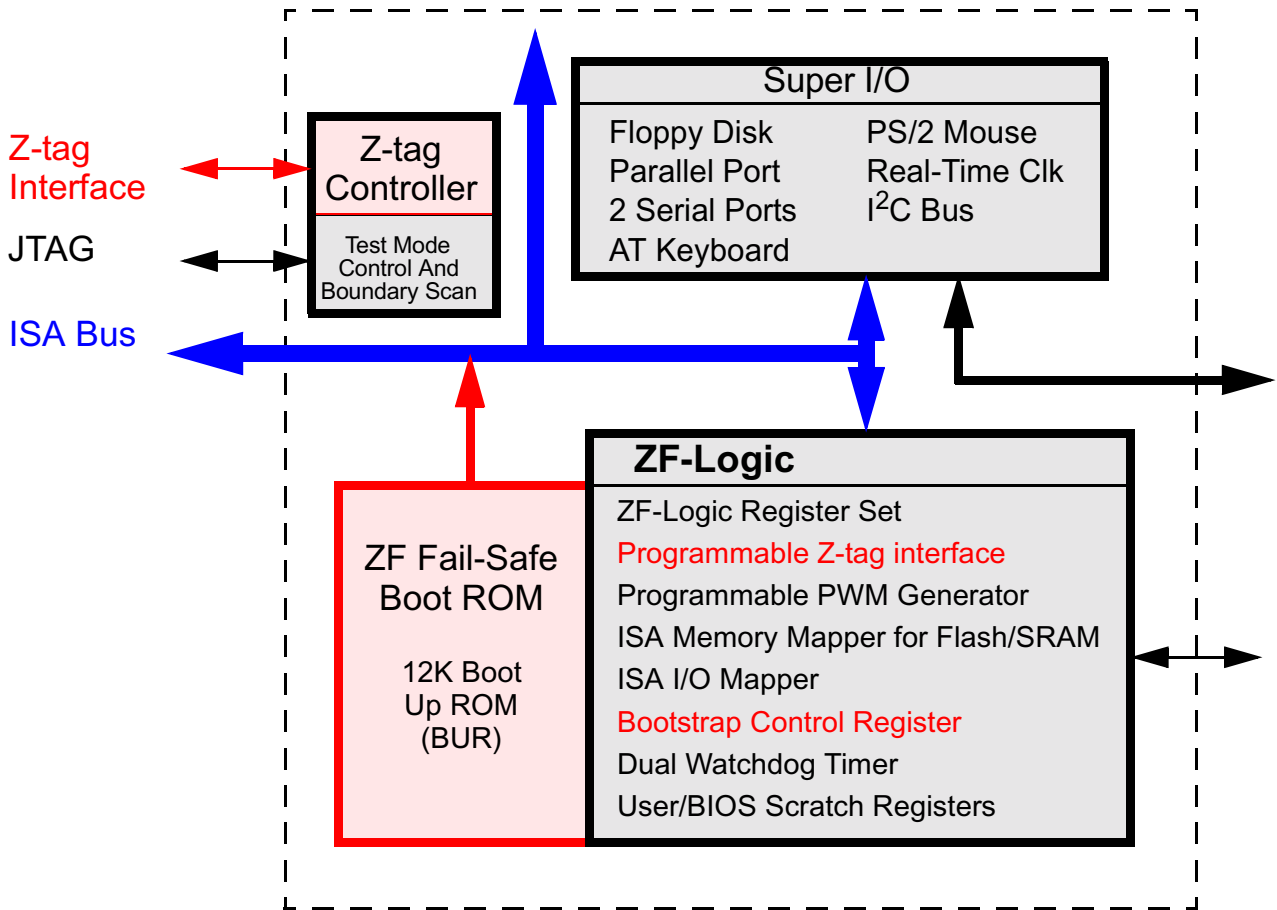
#### 1. First window settings

- 2CH-2BH: Mem\_cs0 window size      0 - FFF000 = 16 MB / 0FFFFH
- 30H-2FH: Mem\_cs0 page              0 - FFF000 = 16 MB / 0
- 28H-27H: Mem\_cs0 base address      0 - FFF000 = 16 MB / F0000H

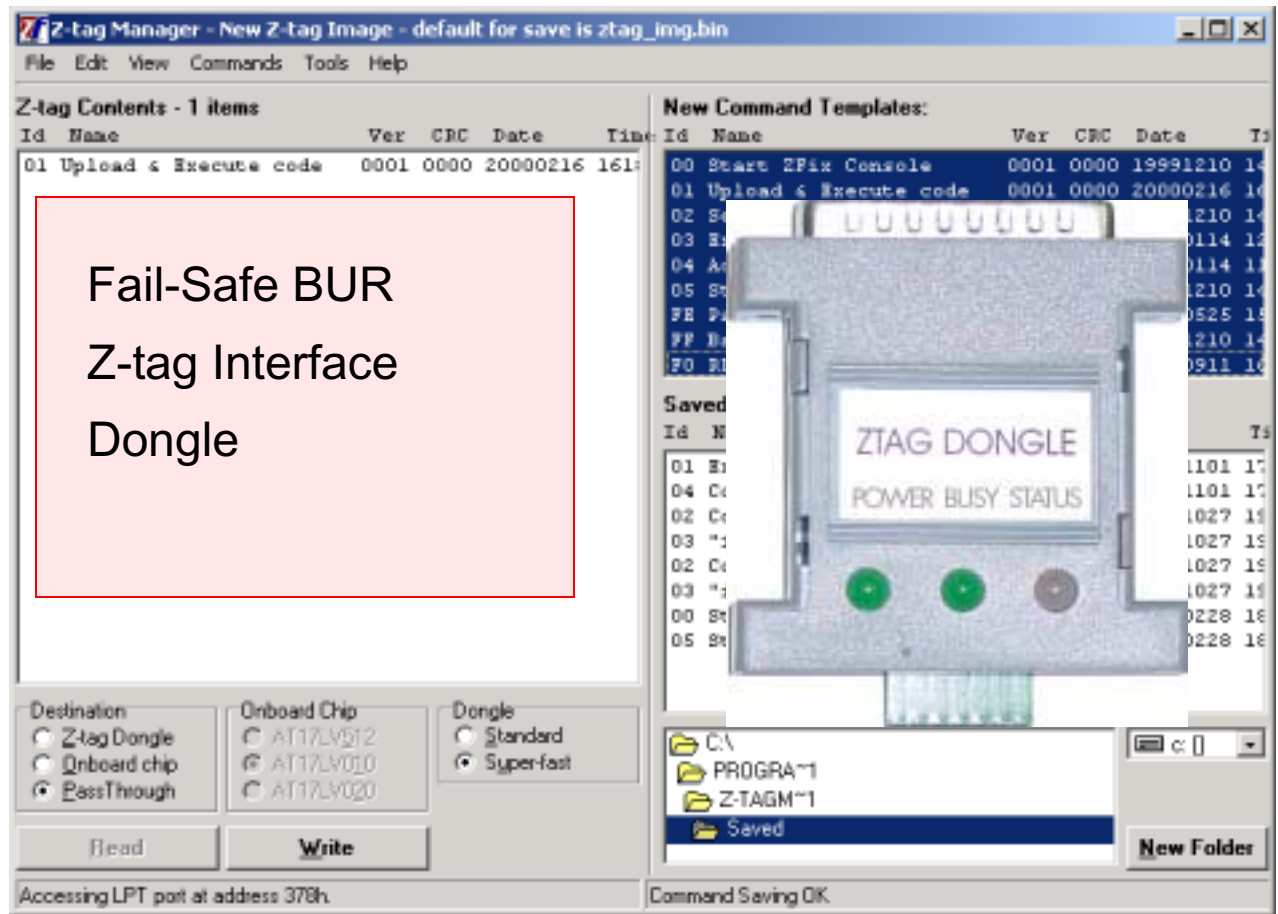
# ZFx86 Training Book

## Chapter 6 - Z-tag and BUR

# 122 ZFx86 Block Diagram



# 123 Using The Z-Tag Manager



Improves speed over using serial interface.

Frees legacy ports from system FLASH update function

Creates a dedicated and simple interface for system upgrading.

## Advantages

- Always Present
- Fast
- Up to 1.5 Mbits/S
- Simple to use
- Automatic
- Easy Configuration
- Initial BIOS Load
- Manufacturing Test
- Diagnostics
- Remote Console
- BIOS Updates
- Application Patches



# 124 | The Z-tag Dongle

## Small size

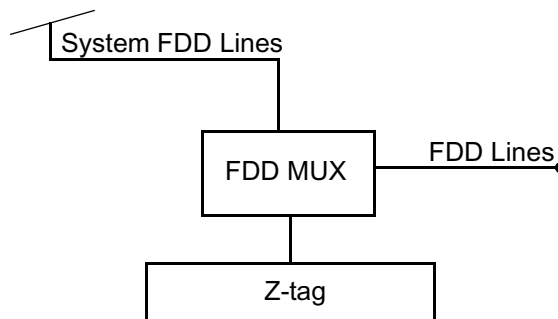
- 14 Pins: 7x2 Dual Row Header
- Use Surface Mount or Through-Hole technology

## Placement

- Anywhere on your board that is convenient
- Ideally at board's edge  
    If access is possible by means of a door or panel, use is simplified



NOTE: Shared pins with floppy interface to eliminate the use of additional pins on the ZFx86



## 125 | The Ztag Dongle (Continued)

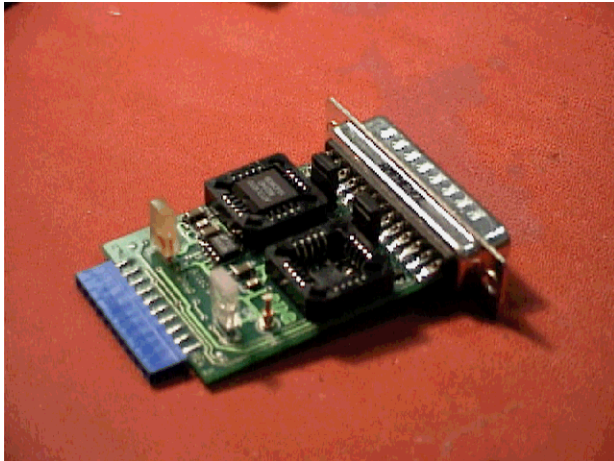
---

Simple inexpensive device.

Facilitates field upgrades

Can store up to 512KB of data

Provides positive feedback to operator



“Memory” Dongle

2 SEEPROMS 256  
KBytes

2 Jumpers

Normal / PassThrough  
Write Protect

2 LEDs

Program



“PassThrough” Dongle

No onboard memory

No jumpers

Fast data transfer

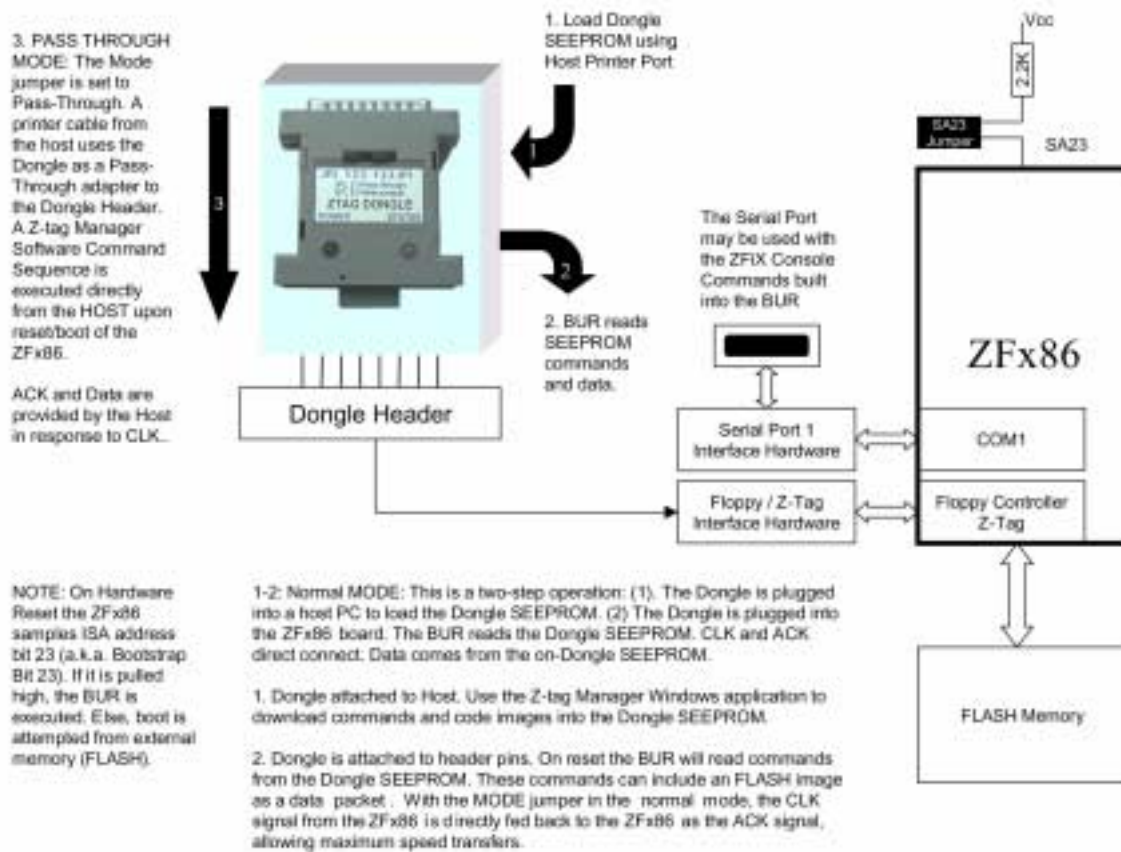
3 LEDs

Power

Busy

Status

# 126 Normal vs. PassThrough Download Mode



PassThrough MODE: Host Connected to Target system directly (using the PassThrough dongle and a printer cable)

Normal MODE: Load the Memory Dongle using the Host system, and carry the Memory Dongle to the Target system

# 127 | The Z-tag Manager Interface

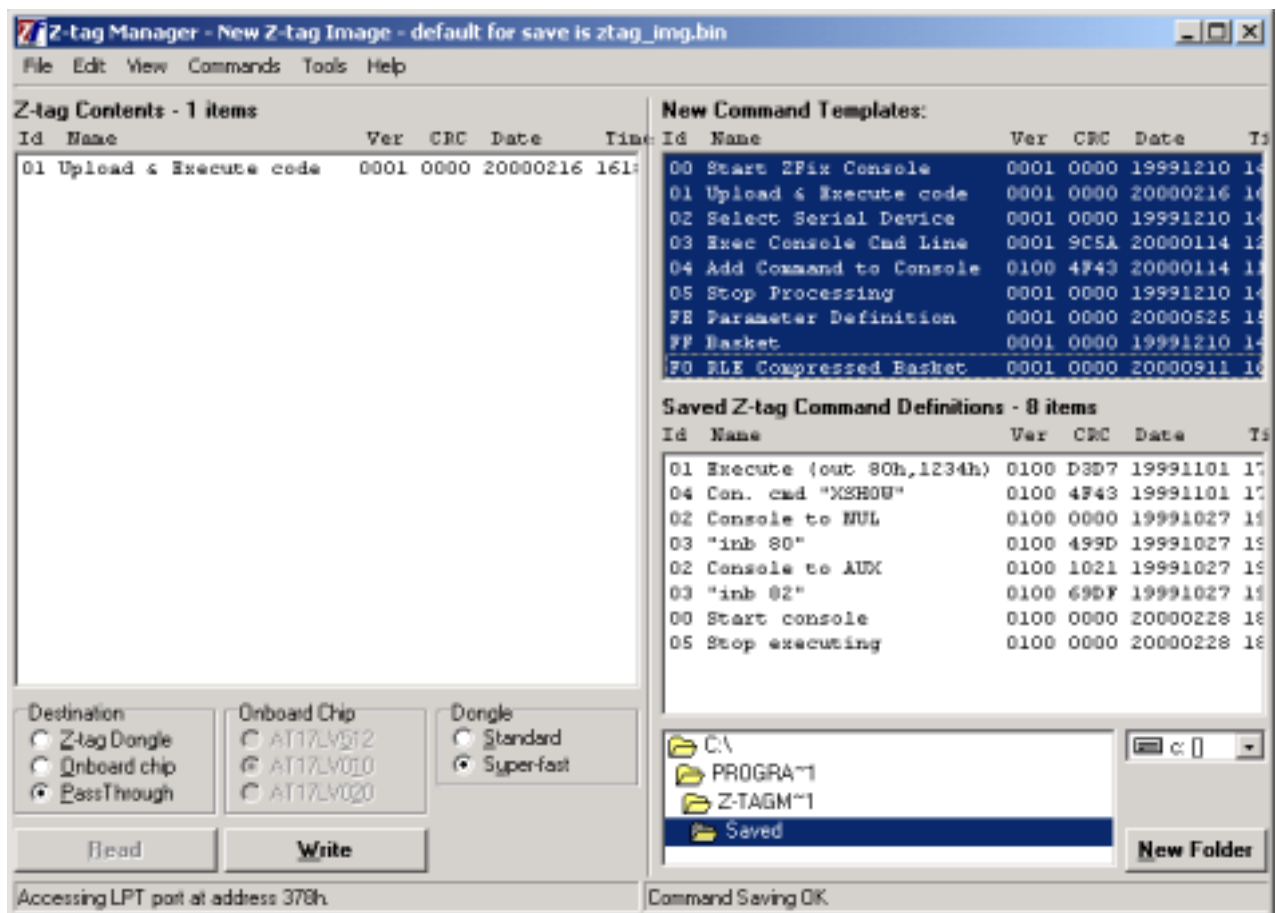
## MS Windows Application

Used to load the Dongle with Data

Simple instructions / Powerful results

Only 7 standard commands

Create your own commands to add special purpose functionality

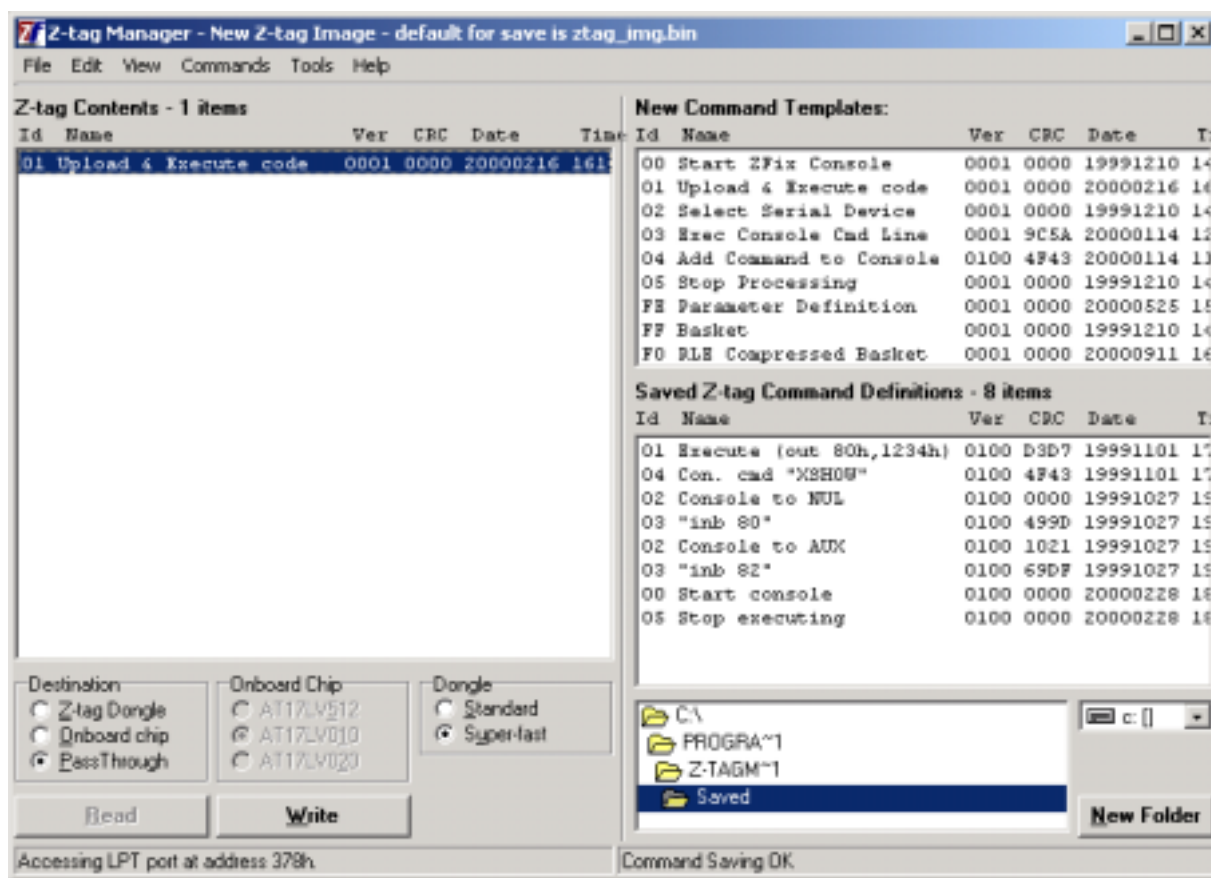


## • Dongle Related

- 01 - Upload and Execute Code
- 05 - Stop
- FF - (Transport Container "Basket" for Data)

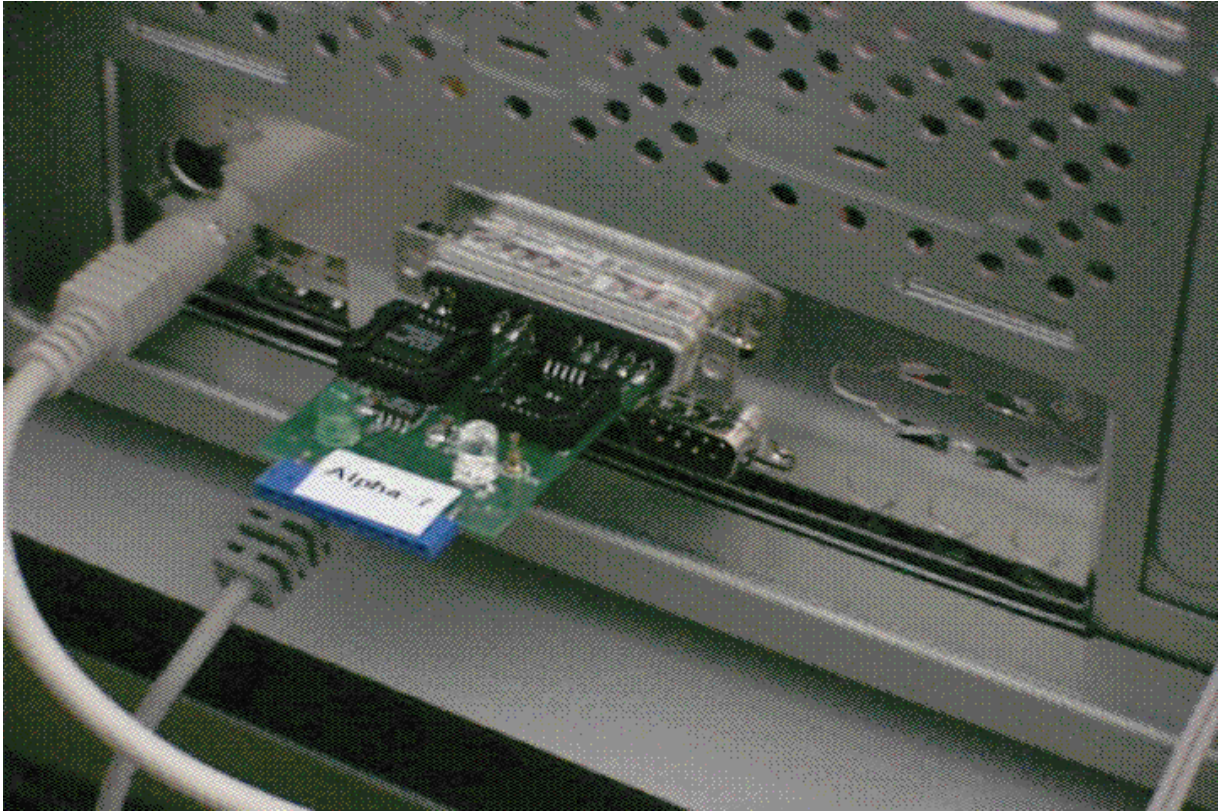
## • Console Related

- 00 - Start/Resume BUR Console
- 02 - Serial Console Mode (toggle)
- 03 - Execute Console Command Line
- 04 - Add Command to Console





# 129 Z-tag "Memory" Dongle Programming



<b>Destination</b> <input checked="" type="radio"/> Z-tag Dongle <input type="radio"/> Onboard chip <input type="radio"/> PassThrough	<b>Onboard Chip</b> <input type="radio"/> AT17LV512 <input checked="" type="radio"/> AT17LV010 <input type="radio"/> AT17LV020	<b>Dongle</b> <input checked="" type="radio"/> Standard <input type="radio"/> Super-fast
<b>Read</b>	<b>Write</b>	
Selected Device is Z-tag		

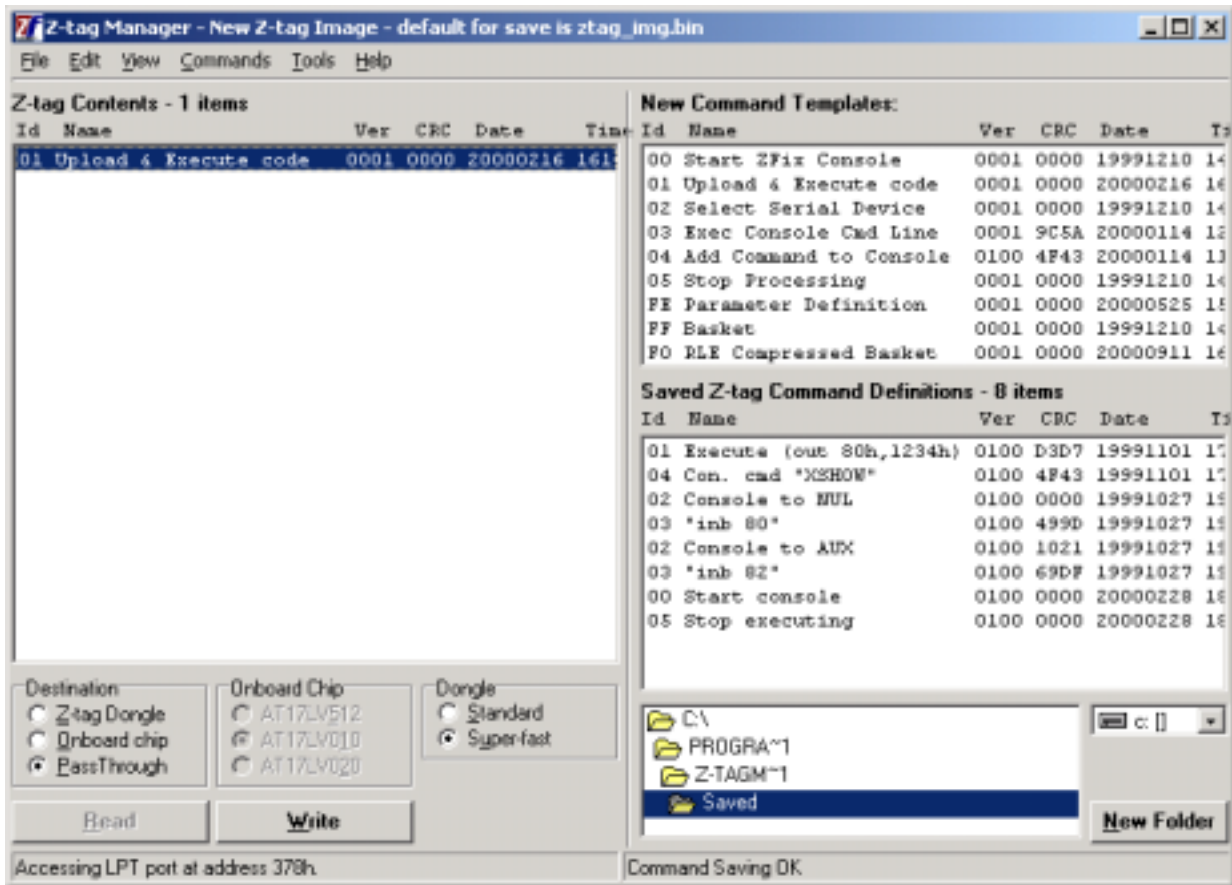
# 130 Z-Tag "PassThrough" Dongle Programming



Destination	Onboard Chip	Dongle
<input type="radio"/> Z-tag Dongle	<input type="radio"/> AT17LV512	<input type="radio"/> Standard
<input type="radio"/> Onboard chip	<input checked="" type="radio"/> AT17LV010	<input checked="" type="radio"/> Super-fast
<input checked="" type="radio"/> PassThrough	<input type="radio"/> AT17LV020	
<input type="button" value="Read"/>		<input type="button" value="Write"/>
Super-fast dongle selected		

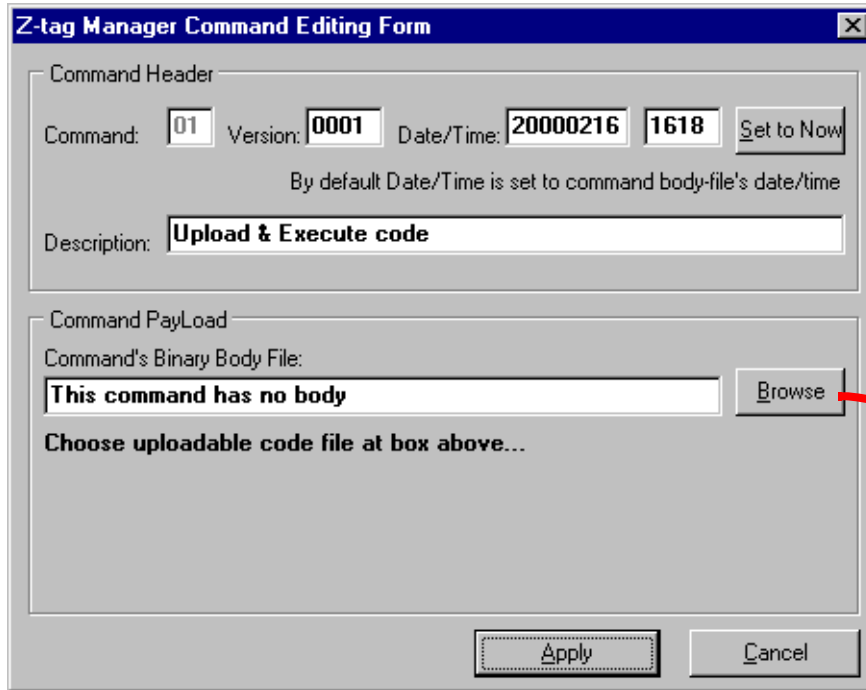
# 131 Put the BIOS in Dongle - First Get Flash Program

Use the "Upload & Execute Code command:





# 132 | Editing Command 01 - Upload & Execute Code



**Z-tag Manager Command Editing Form**

Command Header

Command:  Version:  Date/Time:

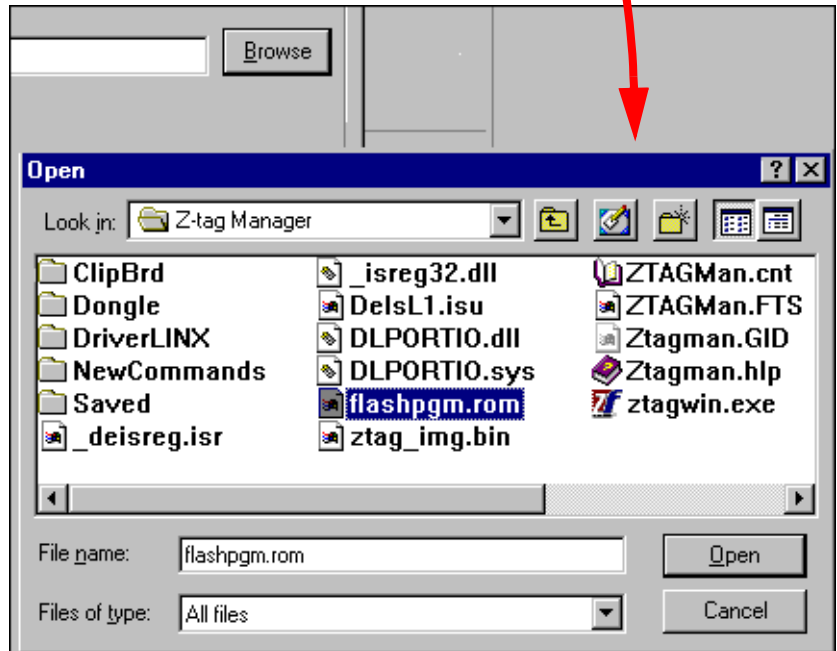
By default Date/Time is set to command body-file's date/time

Description:

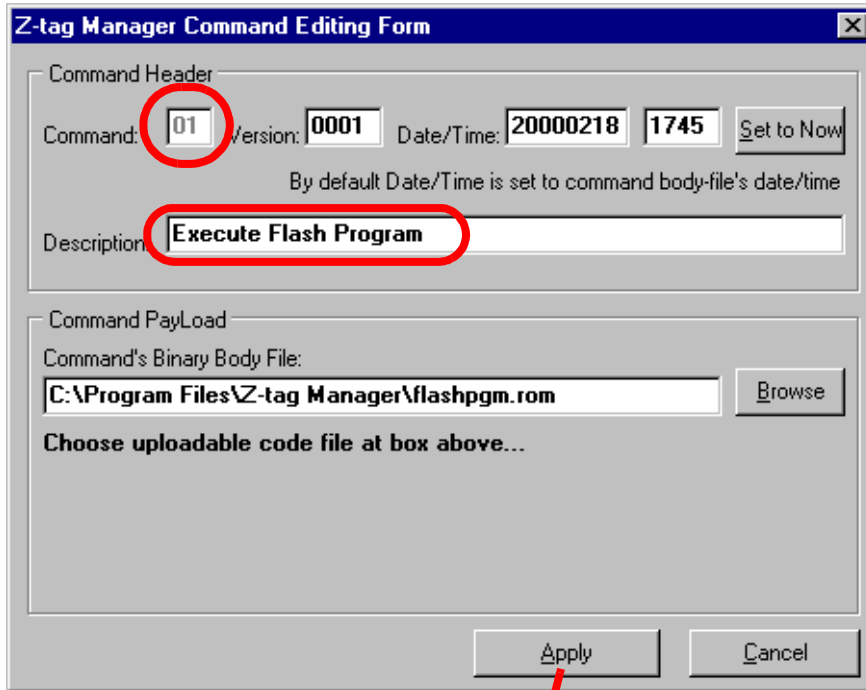
Command PayLoad

Command's Binary Body File:

Choose uploadable code file at box above...



# 133 Edit Selection of Flash Code, Add BIOS Basket



**Z-tag Manager Command Editing Form**

Command Header

Command: **01** /Version: **0001** Date/Time: **20000218** **1745**

By default Date/Time is set to command body-file's date/time

Description: **Execute Flash Program**

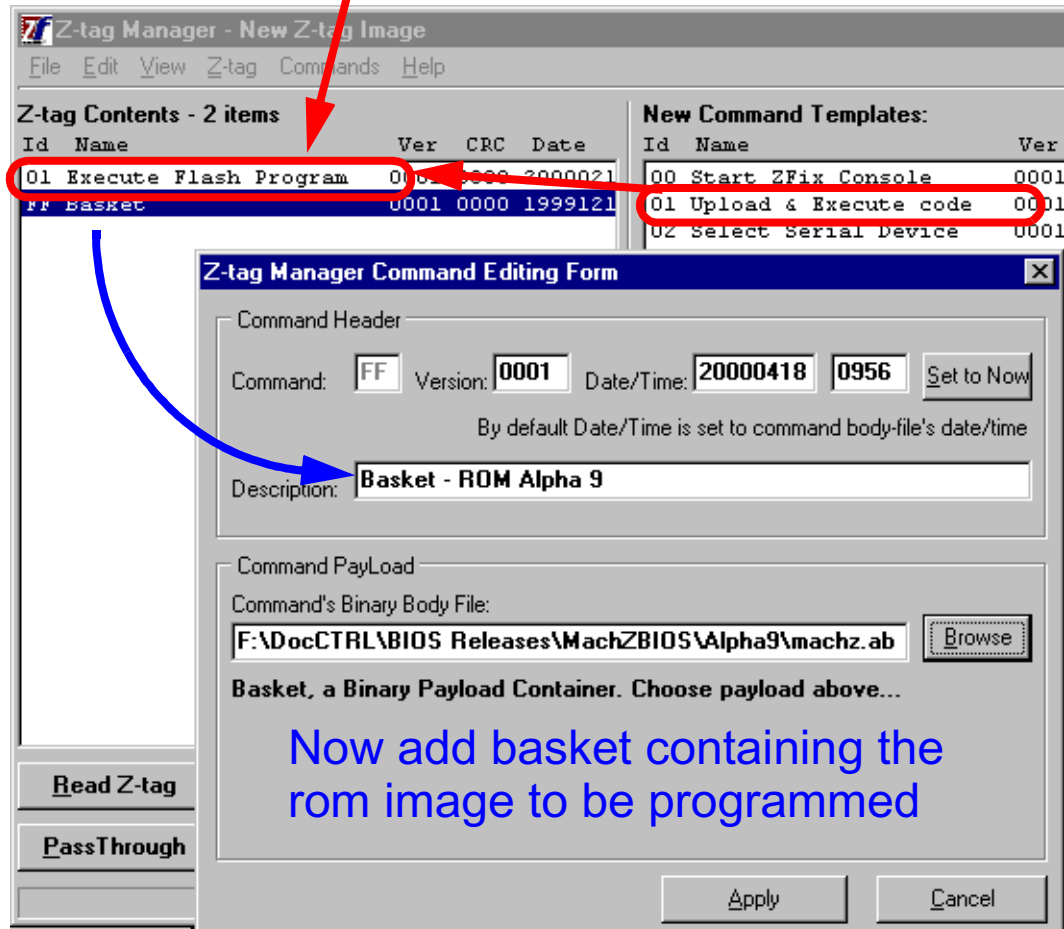
Command Payload

Command's Binary Body File:  
**C:\Program Files\Z-tag Manager\flashpgm.rom**

Choose uploadable code file at box above...

Changed the name of command 01

specified the file which contains the code to program the Flash



**Z-tag Manager - New Z-tag Image**

File Edit View Z-tag Commands Help

Z-tag Contents - 2 items

Id	Name	Ver	CRC	Date
01	Execute Flash Program	0001	0000	20000218
FF	Basket	0001	0000	19991211

New Command Templates:

Id	Name	Ver
00	Start ZFix Console	0001
01	Upload & Execute code	0001
02	Select Serial Device	0001

**Z-tag Manager Command Editing Form**

Command Header

Command: **FF** /Version: **0001** Date/Time: **20000418** **0956**

By default Date/Time is set to command body-file's date/time

Description: **Basket - ROM Alpha 9**

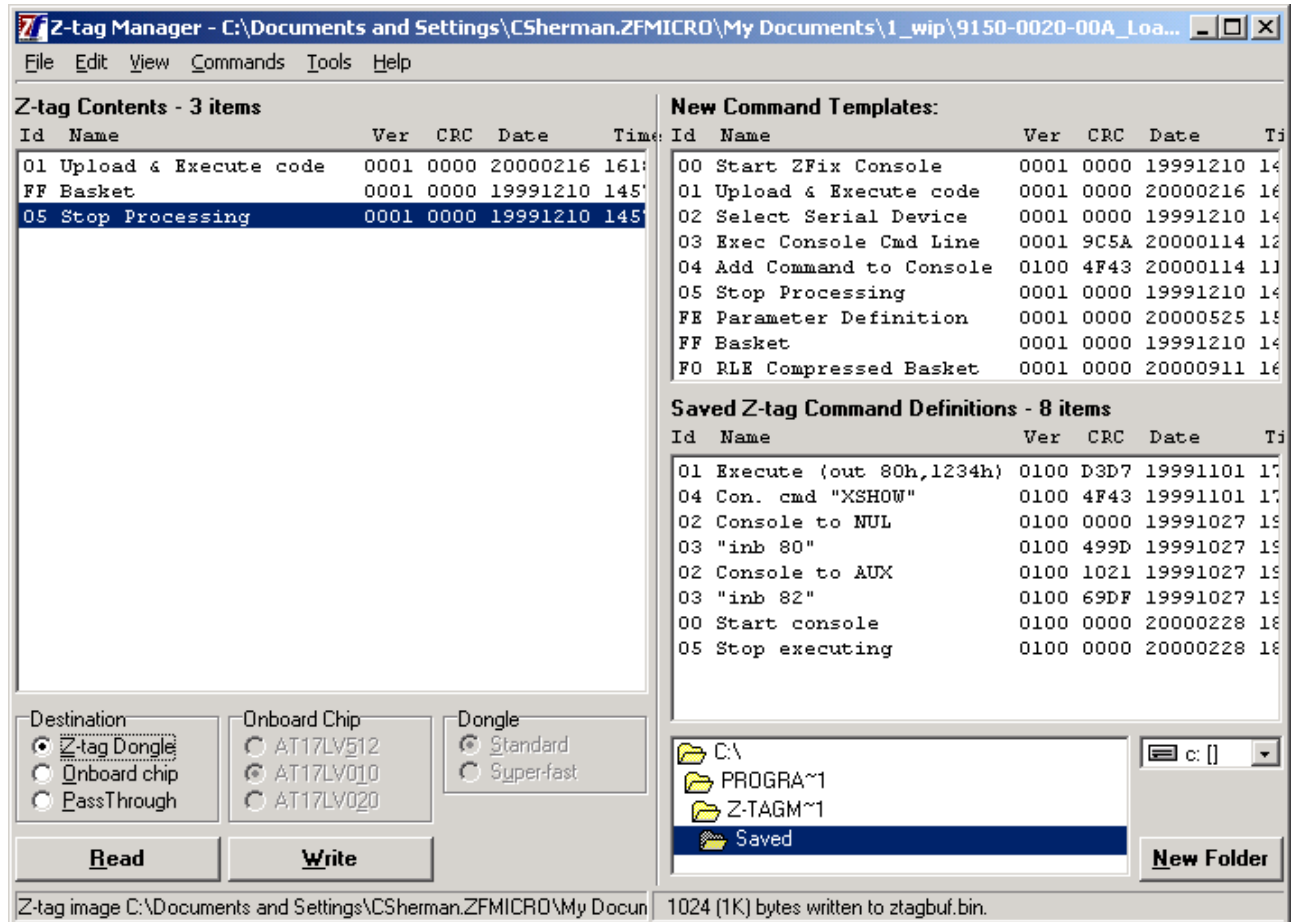
Command Payload

Command's Binary Body File:  
**F:\DocCTRL\BIOS Releases\MachZBIOS\Alpha9\machz.ab**

Basket, a Binary Payload Container. Choose payload above...

**Now add basket containing the rom image to be programmed**

# 134 Add The Stop Command, Create A "Save" Folder



# 135 Copy and Paste Commands to Work Area

**Z-tag Manager - C:\Documents and Settings\CSherman.ZFMICRO\My Documents\1\_wip\9150-0020-00A\_Loa...**

File Edit View Commands Tools Help

**Z-tag Contents - 11 items**

Id	Name	Ver	CRC	Date	Time
02	Select Serial Device	0001	1021	20000609	202
01	AMD programmer	0001	5AD2	20000920	175
FE	BIOS start 1C0000	0001	461E	20010110	103
FF	Phoenix 1.05 BIOS	0001	8C44	20010712	095
01	AMD programmer	0001	5AD2	20000920	175
FE	vxworks loader @ 1A0000	0001	ECB8	20010110	104
FF	vxworks loader romext	0001	10B4	20010515	101
01	AMD programmer	0001	5AD2	20000920	175
FE	vxworks Image @ 100000	0001	0373	20010716	120
FF	vxworks.st_rom	0001	0744	20001214	101
05	Stop Processing	0001	0000	20010807	110

**New Command Templates:**

Id	Name	Ver	CRC	Date	Time
00	Start ZFix Console	0001	0000	19991210	14
01	Upload & Execute code	0001	0000	20000216	16
02	Select Serial Device	0001	0000	19991210	14
03	Exec Console Cmd Line	0001	9C5A	20000114	12
04	Add Command to Console	0100	4F43	20000114	11
05	Stop Processing	0001	0000	19991210	14
FE	Parameter Definition	0001	0000	20000525	15
FF	Basket	0001	0000	19991210	14
FO	RLE Compressed Basket	0001	0000	20000911	16

**Saved Z-tag Command Definitions - 8 items**

Id	Name	Ver	CRC	Date	Time
01	Execute (out 80h,1234h)	0100	D3D7	19991101	17
04	Con. cmd "XSHOW"	0100	4F43	19991101	17
02	Console to NUL	0100	0000	19991027	15
03	"inb 80"	0100	499D	19991027	15
02	Console to AUX	0100	1021	19991027	15
03	"inb 82"	0100	69DF	19991027	15
00	Start console	0100	0000	20000228	18
05	Stop executing	0100	0000	20000228	18

Destination:  Z-tag Dongle  Onboard chip  PassThrough

Onboard Chip:  AT17LV512  AT17LV010  AT17LV020

Dongle:  Standard  Super-fast

Accessing LPT port at address 378h.

File Explorer: C:\ > PROGRAM~1 > Z-TAGM~1 > Saved

# 136 Copy Program Through the PassThrough Dongle

**Z-tag Manager - C:\Documents and Settings\CSherman.ZFMICRO\My Documents\1\_wip\9150-0020-00A\_Loa...**

File Edit View Commands Tools Help

**Z-tag Contents - 11 items**

Id	Name	Ver	CRC	Date	Time
02	Select Serial Device	0001	1021	20000609	202...
01	AMD programmer	0001	5AD2	20000920	175...
FE	BIOS start 1C0000	0001	461E	20010110	103...
FF	Phoenix 1.05 BIOS	0001	8C44	20010712	095...
01	AMD programmer	0001	5AD2	20000920	175...
FE	vxworks loader @ 1A0000	0001	ECB8	20010110	104...
FF	vxworks loader romext	0001	10B4	20010515	101...
01	AMD programmer	0001	5AD2	20000920	175...
FE	vxworks Image @ 100000	0001	0373	20010716	120...
FF	vxworks.st_rom	0001	0744	20001214	101...
05	Stop Processing	0001	0000	20010807	110...

**New Command Templates:**

Id	Name	Ver	CRC	Date	Time
00	Start ZFix Console	0001	0000	19991210	14...
01	Upload & Execute code	0001	0000	20000216	16...
02	Select Serial Device	0001	0000	19991210	14...
03	Exec Console Cmd Line	0001	9C5A	20000114	12...
04	Add Command to Console	0100	4F43	20000114	11...
05	Stop Processing	0001	0000	19991210	14...
FE	Parameter Definition	0001	0000	20000525	19...
FF	Basket	0001	0000	19991210	14...
F0	RLE Compressed Basket	0001	0000	20000911	16...

**Saved Z-tag Command Definitions - 8 items**

Id	Name	Ver	CRC	Date	Time
00	D3D7	19991101	17...		
00	4F43	19991101	17...		
00	0000	19991027	19...		
00	499D	19991027	19...		
00	1021	19991027	19...		
00	69DF	19991027	19...		
00	0000	20000228	18...		
05	Stop executing	0100	0000	20000228	18...

**Z-tag Manager is busy...**

Writing parallel passthrough data... Close/Stop

Destination:  Z-tag Dongle  Onboard chip  PassThrough

Onboard Chip:  AT17LV512  AT17LV010  AT17LV020

Dongle:  Standard  Super-fast

Read Write

Super-fast dongle selected

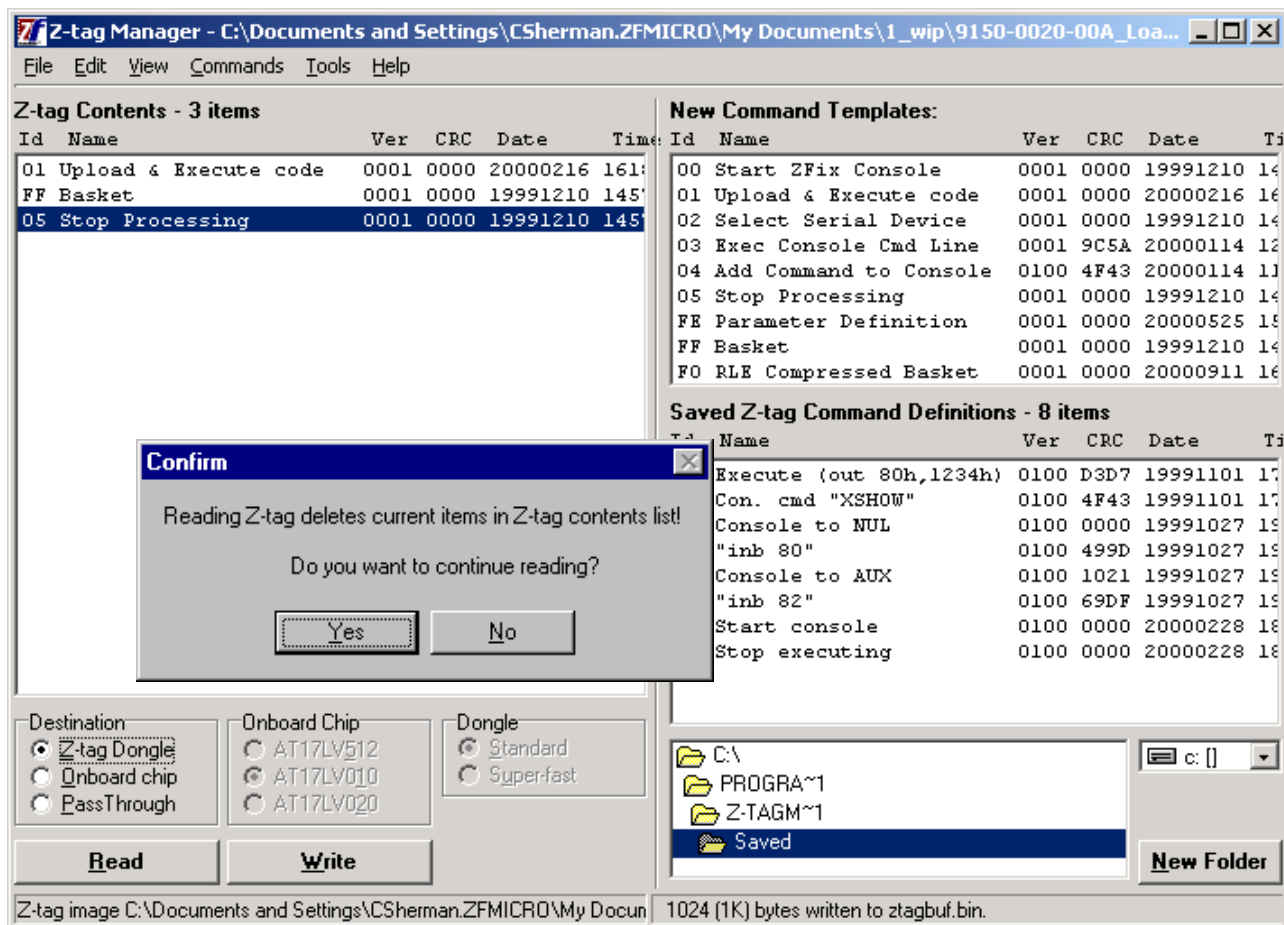
680960 (665K) bytes processed.

C:\

- PROGRA~1
- Z-TAGM~1
- Saved

New Folder

# 137 Test by Reading Back from the “Memory” Dongle



The program is stored in an allocated buffer when you read it, not over your saved commands.

Note that you cannot “Read Back” your Z-tag contents using the PassThrough dongle.

# 138 BUR Version Test Program Source Code

```
;; Copyright 2002 ZF Micro Devices, Inc. All rights reserved.
title ZFx86 BETA Code sample Obtains BUR Version Number

; build statements:
; ml /F1 burver1.asm
; exe2com burver1 0 (this routine available on ZF web site)

.486
burrom          segment USE16 at 0f000h
; Services table. These are the function pointers for
; uploaded code use.
                org      0ff00h ; f000:ff00 in BUR ROM
Bur_Version     db        ?

                org      0ff0ah
CRLF            label    far ; call ff00:ff0a --> CR/LF to COM1
                org      0ff22h
SerOut16       label    far ; call ff00:ff22 --> AX to COM1 as decimal
                org      0ff3ah
SerSend        label    far ; call ff00:ff3a --> Charout to COM1
burrom          ends

CODE            segment USE16 'CODE'
                assume   cs:code

START:

                push     cs
                pop      es
                mov      di,offset VerText ; ES:DI - text to show
                xor      cx,cx ; display until 0 reached
                call     SerSend

                les      bx,psBur_Version ; es:bx --> Bur_Version String
                mov      ax,es:[bx]

                call     SerOut16 ; display AX to COM1 as decimal
                call     CRLF ; CR/LF to COM1

                retf ; resume with BUR

psBur_Version   dd      Bur_Version ; define string pointer
VerText        db      'BUR Version: ',0

CODE            ends
                end      START
```

## 139 BUR (Fail-safe) Boot Up ROM

---

BUR is built-in software that serves as a prototype debug tool and Flash update utility. BUR is an internal 12K binary ROM image.

Functionality can be divided into four categories:

- Basic component initialization
- Elementary debugger console functionality through COM1
- Data fetch and execution through Z-tag interface
- Basic OS functionality for user code

Uses of the BUR

- Manufacturing/field tool
- Debug tool
- Fail-safe System



## 140 Basic Component Initialization

---

After initialization, following system components are active:

- North Bridge
- South Bridge
- ISA Bus
- Internal Static RAM
- IRQ controller
- Timer (8259)
- COM1
- Z-tag interface

# 141 Elementary debugger console functionality

## ZFix Console Commands

Command	Action
i[n[b]]/inw/ind <port>	read 8/16/32-bit value from port
o[ut[b]]/outw/outd <port> <value>	write 8/16/32-bit value to port
zfr <register>	read 8-bit value from ZFLogic register
zfw <register> <value>	write 8-bit value to ZFLogic register
db/dw/dd <address>	display memory in byte/word/dword mode
d	display next memory page in previous mode poke[b]/pokew/poked <address> <value(s)> -
linear	use linear mode addressing
real	use real mode addressing
h[elp]/?	show help
ver	display version information
speed <96/19/38/56/115> <hs>	serial speed. Set hs to 1 for RTS/CTS <sup>a</sup>
yload <address>	load data through YModem to address
yload <address> <length> [filename]	send data through YModem from address
g[o] <address>	start executing from address
dls	Display available download segment address

a. The default speed on power up is 9600. The <hs> handshake bit is currently not working. You may try higher speeds, but you may lose data.

```
ZFiX - ZFx86 PCe Internal Console
(c)2001 ZF Micro Devices, Inc.
: DLS
0070 // Display available segment for
: yload 70:0 downloads
Please start YModem transmission now or press <ESC> ...
CCCCCCCCCCCC ← waiting for transfer
YModem data transfer succeeded.
: g 70:0
BUR Version: 0101
```

## 142 | Data Fetch and Execute

---

There are 7 different type of commands which the BUR understands and executes:

- 00 - Start/Resume BUR console.
- 01 - Upload and execute code.
- 02 - Serial console mode.
- 03 - Execute console command line.
- 04 - Add command to a console.
- 05 - Stop.
- FF - Basket

# 143 Basic OS functionality for user code

```
;; Copyright 2002 ZF Embedded, Inc. All rights reserved.
title ZFx86 BETA Code sample Obtains BUR Version Number

; build statements:
; ml /Fl burver1.asm
; exe2com burver1 0 (this routine available on ZF web site)

.486
burrom          segment USE16 at 0f000h
; Services table. These are the function pointers for
; uploaded code use.
                org      0ff00h ; f000:ff00 in BUR ROM
Bur_Version     db        ?

                org      0ff0ah
CRLF            label    far ; call ff00:ff0a --> CR/LF to COM1
                org      0ff22h
SerOut16        label    far ; call ff00:ff22 --> AX to COM1 as decimal
                org      0ff3ah
SerSend         label    far ; call ff00:ff3a --> Charout to COM1
burrom          ends

CODE            segment USE16 'CODE'
                assume   cs:code

START:

                push    cs
                pop     es
                mov     di,offset VerText ; ES:DI - text to show
                xor     cx,cx ; display until 0 reached
                call    SerSend

                les     bx,psBur_Version ; es:bx --> Bur_Version String
                mov     ax,es:[bx]

                call    SerOut16 ; display AX to COM1 as decimal
                call    CRLF ; CR/LF to COM1

                retf ; resume with BUR

psBur_Version   dd        Bur_Version ; define string pointer
VerText         db        'BUR Version: ',0

CODE            ends
                end     START
```

## 144 Manufacturing and Field Tools

---

Manufacturing cycle can use the dongle at two stages:

- Load the Manufacturing Test program to allow diagnostic exercising of the device.
- Update the device as it goes out the door

Very inexpensive device to provide to the field personnel to troubleshoot on-site.

## 145 | BUR: Debug tool

---

During initial bring-up the BUR allows designer to not populate any other devices on the board with exception of the power and clock circuits.

ZFx86 DRAM configurations were all debugged using the BUR code only.

Small applications can be brought in to the device and always executed from Z-tag interface

## 146 | Fail-safe System

---

Uses the Z-tag interface, dual WDT, and BUR Code to provide automatic recovery of the system.

One recovery scenario:

Application overwrites a portion of the OS with Watch Dog enabled.

Application hangs due to lack of OS.

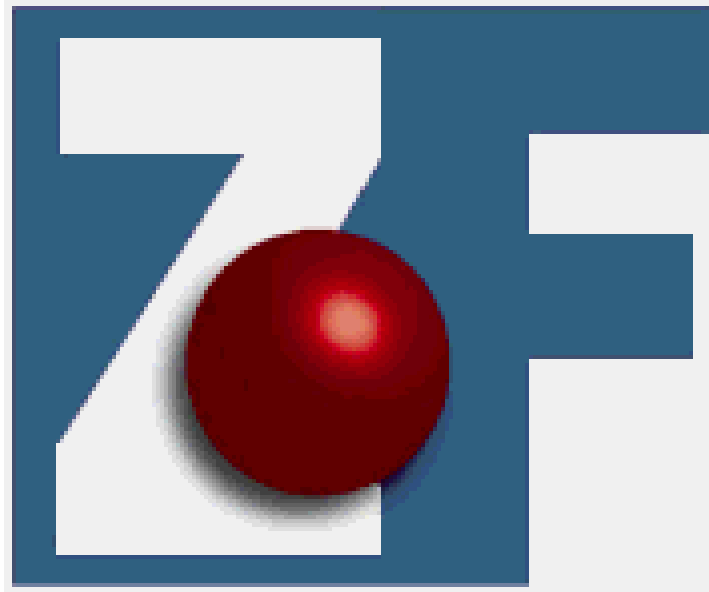
WDT re-boots into BUR code (bit 23 tied high)

Bur code boots, finds Z-tag device on port and loads through the port the information present in the serial EPROM on board.

The code present in the serial EPROM includes the CRC for the flash device. The BUR code then calculates the flash CRC and compares it to the expected value.

If CRC is bad, additional programs can be downloaded through the Z-tag interface to execute a modem call to the factory requesting new code.

Once new code is downloaded, the flash is updated and a system reset is executed (a South Bridge Reset or a jump to ffff fff0) after setting the BUR base address to zero in index register 57H/58H.



**ZF Micro Devices, Inc.**

**1052 Elwell Court**

**Palo Alto, California 94303**

**(650) 965-3800 · Fax 965-4050**

**[www.zfmicro.com](http://www.zfmicro.com)**