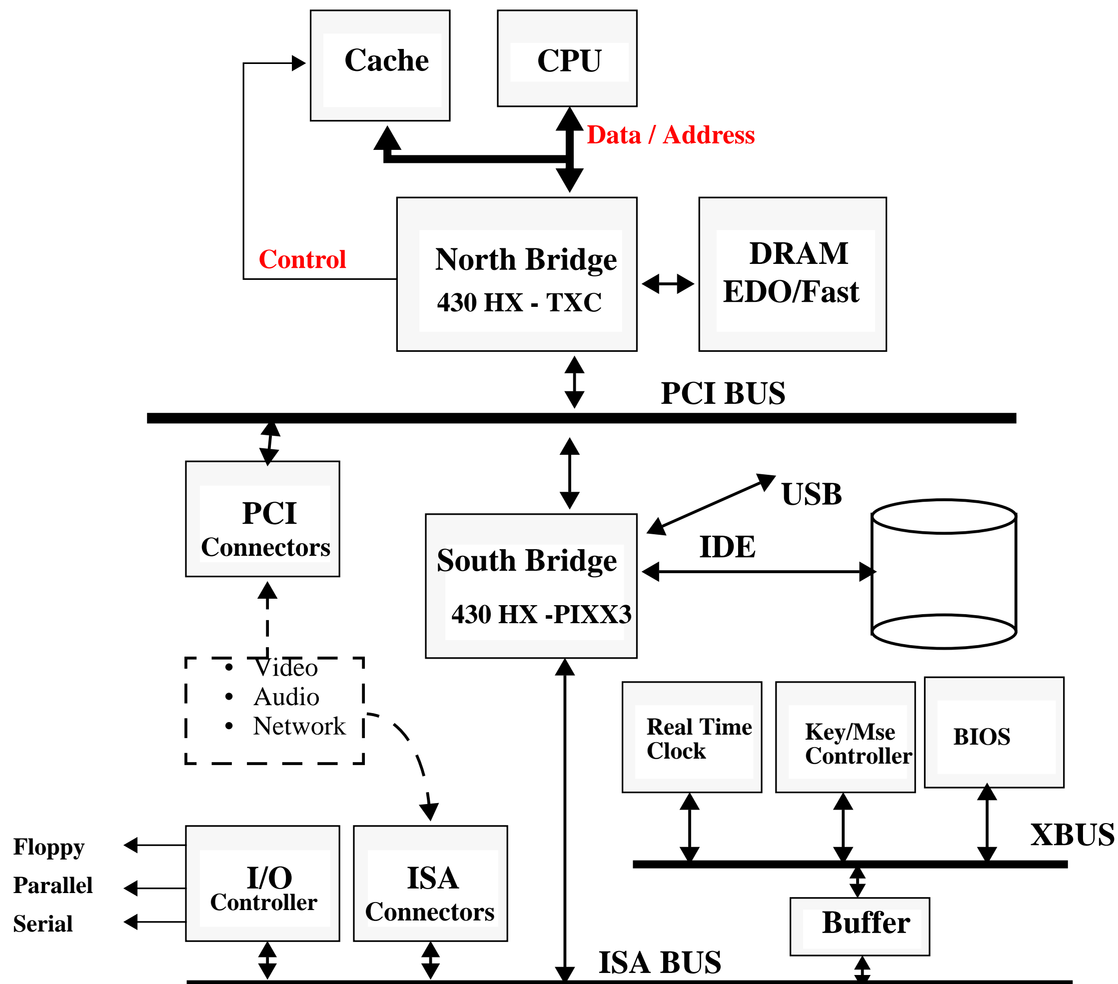
Smart Port Card (SPC)

William Eatherton
Toshiya Aramaki
Edward Spitznagel
Guru Parulkar
Applied Research Lab
Washington University in St. Louis

Design Goals

- For several Gigabit Switch related projects, a processing element attached to some or all of the ports would be an excellent platform for experimentation.
- The simplest solution would be a bank of PC's with one attached to each port of the switch via an APIC and ribbon cable,
  - Primary disadvantages is cost and space requirements
  - Also this "system" could be difficult to maintain since reset events and initialization are not necessarily synchronized
- NetBSD is the target OS, the goal is to modify the kernel to operate in an embedded system without non-volatile storage (flash memory or hard disk) and the only I/O being the ATM network connection.
  - To accomplish the above goal, the OS and applications are loaded over the ATM network
- To reduce debugging time and simplify upgrades to future versions of NetBSD, one goal is to modify the run time NetBSD kernel as little as possible.
  - An attempt will be made to limit kernel customization to boot loader code and the standard kernel config file
  - To accomplish this goal, it is necessary to instantiate or emulate system resources required by the kernel for normal operation (RTC, interrupt controller, timer, etc.)

Architecture of standard 430HX Pentium Motherboard



- CPU: Any 3V Pentium
- Cache: 256K / 512K, Pipelined burst SRAM
- PIIX3: Contains DMA Controller, Programmable Interrupt Controller (PIC), Programmable Interval Timer (PIT), Reset Logic
- Real Time Clock (RTC) also contains BIOS configuration data in “NVRAM”, *the chip is battery powered*

CPU Accesses to Memory Space

- · DRAM Memory from 1 MB to a maximum of 512 MB, this area is ALWAYS cacheable and not forwarded to PCI bus
- PCI Memory space from the top of DRAM to (4 GB - 512 KB), this area is NEVER cacheable, and all memory accesses forwarded to PCI bus
- DOS Area (00000-A0000h),(640KB), always cacheable
- Expansion Area (C0000-DFFFFh) This 128-KB area is divided into eight 16-KB segments. Each segment can be assigned Cacheable/Non-cacheable and DRAM/PCI access.
- The 128-KB BIOS memory space is located at 000E0000–000FFFFFh (top of 1 Mbyte) and is aliased at FFFE0000h (top of 4 GB).
  - Extended System BIOS Area (E0000-EFFFFh) This 64-KB area is divided into four 16-KB segments. Each segment can be assigned independent cacheability, read, and write attributes.
  - System BIOS Area (F0000-FFFFFh) This area is a single 64-KB segment. This segment can be assigned cacheability, read, and write attributes.
- Other areas like video RAM

PCI TO PIIX3 Memory accesses

- All PCI memory accesses in the lower 16MB or the upper 512KB of 32-bit memory space, that are not positively decoded, will be subtractively decoded by the PIIX3 chip
- The 128-KB System BIOS memory space is located at 000E0000–000FFFFFh (top of 1 Mbyte) and is aliased at FFFE0000h (top of 4 GB).
- The additional 384-KB “Extended BIOS region” resides at FFF80000–FFFDFFFFh

I/O and Configuration Space Access

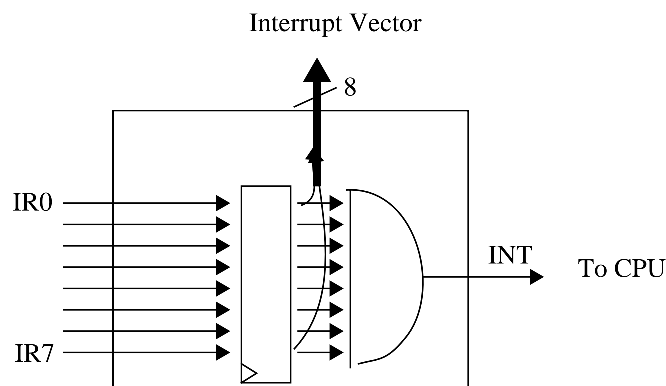
- PC has 64KB of I/O space,
- The North bridge, South bridge, and all PCI devices have a dedicated configuration space that is not a distributed space
- The CPU uses the same address and data lines for I/O accesses as memory accesses but has a dedicated signal that indicates which one is being accessed
- The North Bridge has 2 I/O registers, Config Address(0x0cf8) and Config Data (0x0xfc). I/O space accesses to any other address are passed to the PCI bus
- All configuration of the North Bridge, South Bridge, and PCI devices are done through a "window" accessed by the above 2 I/O space registers
- On the PCI bus, a 4-bit "command" field contains information indicating if access is to memory space, configuration space, or I/O space
- The PIIX3 chip positively decodes I/O accesses to ISA legacy registers:
  - Programmable Interval Timer (PIT)
  - Programmable Interrupt Controller (PIC)
  - Non Maskable Interrupt control (NMI)
  - Reset control
  - RTC & NVRAM RAM

Hardware Interrupts

1 Maskable Interrupts

* Standard PC systems have 2 cascaded PIC chips (or PIC cores inside system chips like the Piix3) providing 15 interrupts lines
* The PCI bus uses 4 of these interrupt lines for its 4 shared interrupts (INTA-INTD)
* PIC chips/cores like the 8259A have much more programmability than is ever used in a standard PC system, 4 8-bit registers set up by the BIOS control the PIC operation
* PIC options include Master/slave, edge/level interrupt, etc.

• Basic view of PIC chip/core:



2 Non-Maskable Interrupts

* These interrupts result from non-recoverable system failure
* Causes of Non-Maskable interrupts are: ISA Bus system errors (signalled by assertion of IOCHK), memory parity errors (signalled via assertion of SERR by North Bridge), and PCI Bus errors also signalled by assertion of the SERR PCI signal,.
* Non-Maskable interrupts are signalled to the CPU by the Pixx3 or stand alone PIC chip via the NMI pin.

PC Initialization for standard 430HX system with the NetBSD OS

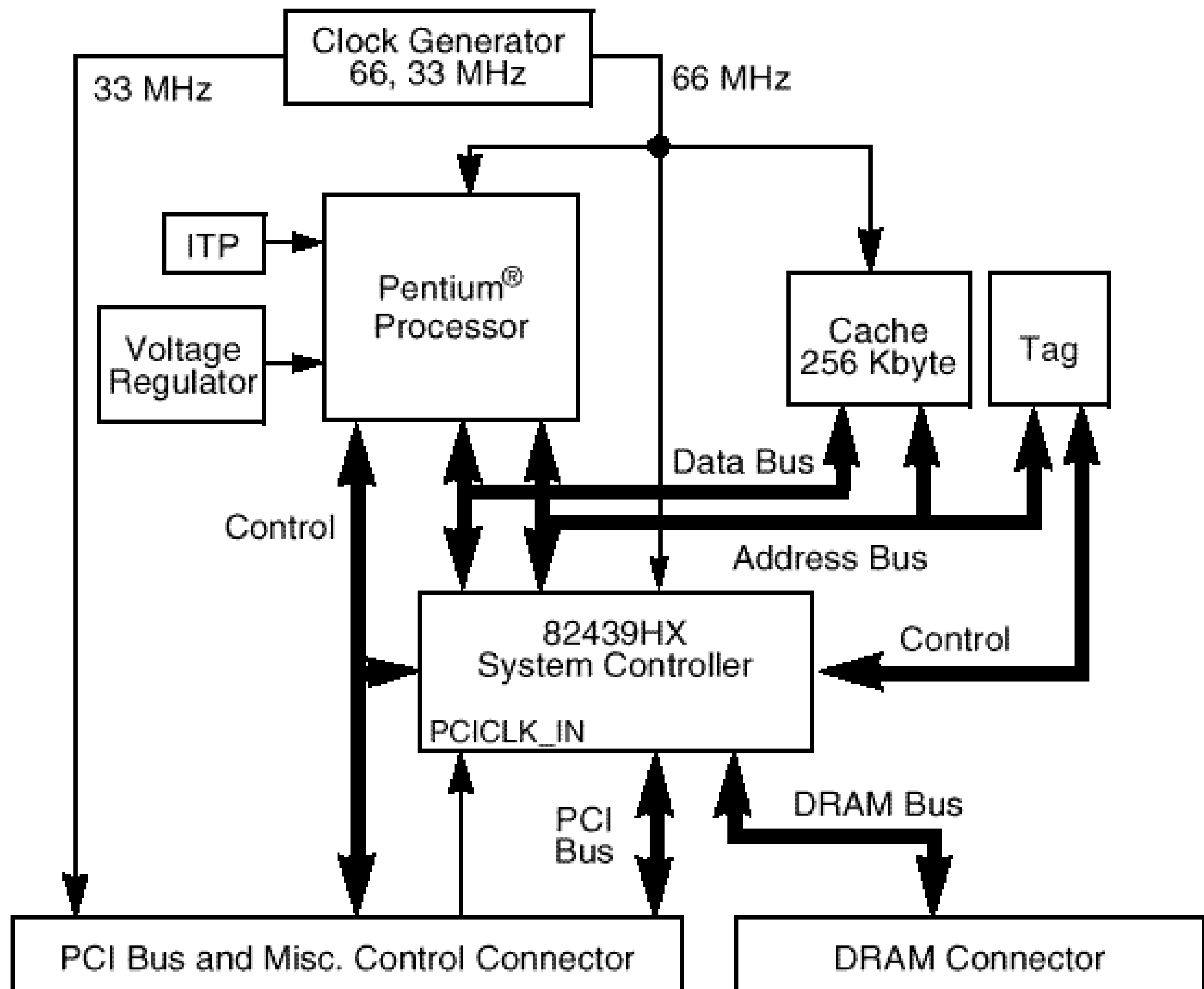
1. At power up or after a hard reset, the signal PWROK transitions (via analog reset circuitry) from deasserted to asserted. The PWROK signal enters the PIIX3 chip which with appropriate timing then deasserts the CPU and PCI reset signals.
2. Pentium does internal initialization and then does a memory read from address 0xfffffff0. At reset, the North Bridge chip has this memory area marked as disabled so read is passed to PCI bus.
3. At reset all PCI devices are disabled so the memory access will be subtractively decoded by the PIIX3 chip. The PIIX3 chip will know this is a BIOS read request and wiggle signals so that the appropriate data from the BIOS is passed back to the PCI bus, which passes the data to the CPU.
4. The CPU executes this instruction, which is usually a jump to the top half of the normal BIOS memory space 0xffff0000 (since we are 16 bytes from the end of memory).
5. At this point the BIOS code instructs the pentium to copy the System BIOS code to the memory locations 0x000E0000–0x000FFFFF, and jumps to the following instruction but in the aliased copy of the BIOS code under 1 megabyte.
6. At this point the Power On Self Test (POST) portion of the BIOS is run. At this point the amount of physical memory is checked, PCI devices are configured and enabled, A few registers are written in the North and South Bridges, etc.
7. The BIOS Data Area is created at memory location 0040:0000h (000400h) with a size of 256 bytes (0040:0000h-0040:00FFh). This area contains information about the system configuration. This data area has information on serial ports, parallel ports, floppy drives, hard drives, graphic adapters, memory size, keyboard specifics, math co-processor, mouse
8. At power-on, POST creates a table of interrupt vectors, pointers to the entry point of interrupt service routines, with a size of 1024 bytes, and contains a maximum of 256 interrupt vectors (of 4 bytes). The interrupt vector table is located from memory location 0000:0000h up to 0000:03FCh.
9. When the BIOS code is done, it calls the BIOS interrupt #19, which reads Sector1, Track 0 (512 bytes) of the "boot disk" into memory at location 0000:7C00H (This is the MBR).
10. If location 0000:7DFEH (last two bytes of that sector) contains the value AA55H, the BIOS bootstrap routine will transfer control to 0000:7C00H. Otherwise, the

PC Init Continued

boot code in that sector is bad and the boot routine then calls INT18 which can be loaded by expansion ROMs to do things like load an OS over ethernet using TCP/IP.

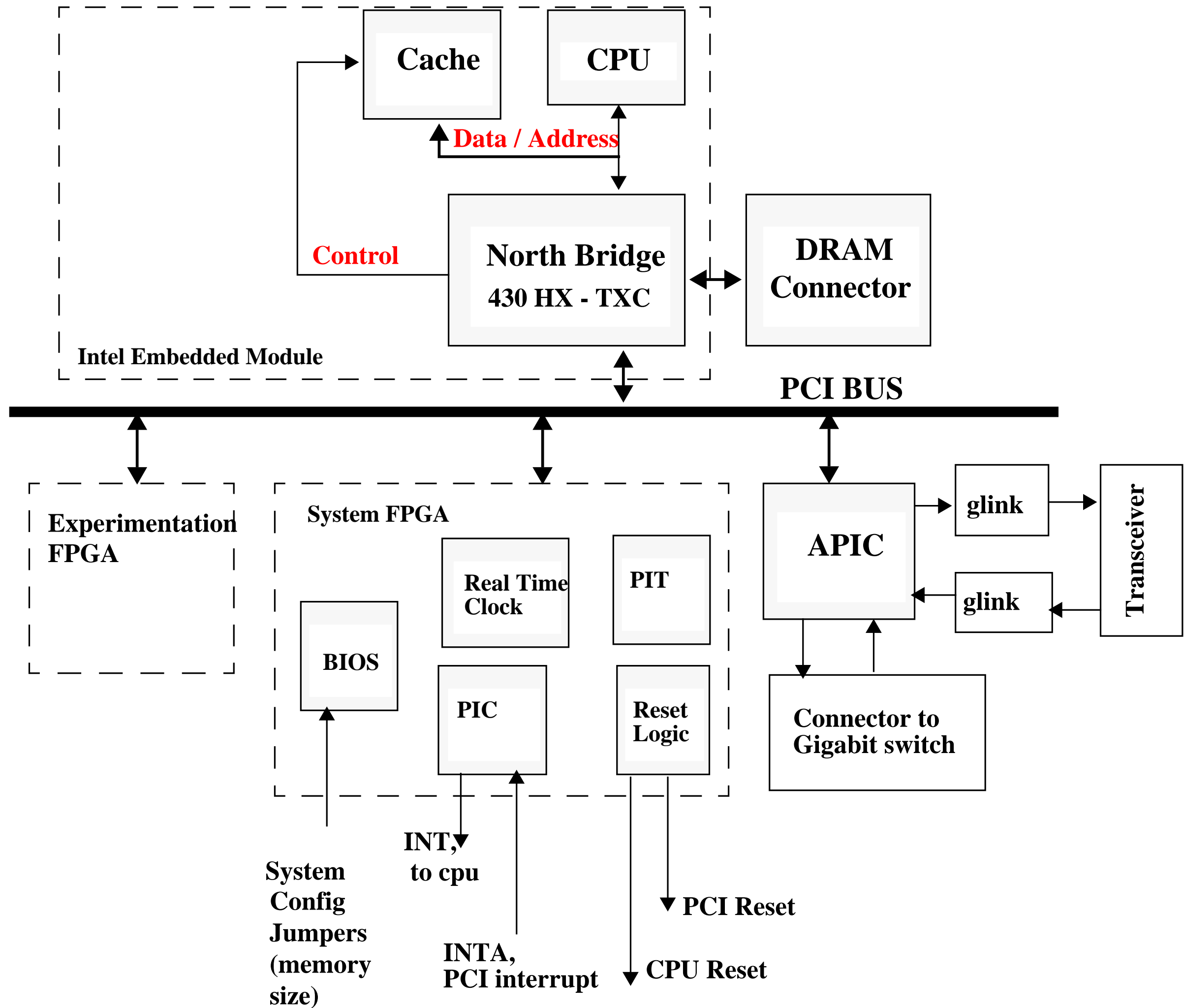
11. BIOS executes the code in the MBR... this is usually the boot selection program (i.e. OS-BS). this program uses the BIOS for I/O since it is limited in size to about 480 bytes or so.
12. the boot selector looks at the MBR to find the start of the selected "fdisk" partition and uses the BIOS to load the first sector of that "fdisk" partition into RAM
13. the boot selector executes the first sector of the "fdisk"partition. under BSD this is the file /usr/mdec/sdboot.
14. sdboot is limited in size to one sector (512 bytes). it uses the BIOS calls to load in the second level bootstrap from the disk. this bootstrap is located on the next 31 sectors (and is thus limited to 15872 bytes of code/data).this is the file /usr/mdec/bootsd.
15. sdboot executes bootsd also known as Boot 2. bootsd uses the BIOS to prompt the user on which kernel to load.
16. either the user enters a kernel name, or there is a time-out. Bootsd then uses BIOS I/O to read the unix file system and load a kernel into RAM. (so it has to understand the filesystem format). Since BIOS runs in REAL mode, and Boot2 runs in protected mode, Boot 2 switches back to real mode just before each BIOS call and then back to protected mode on each return (Ugly Code!!!!).
17. the bootsd program executes the kernel (passing in the boot flags [i.e. single user boot, etc.]).
18. the kernel inits memory management, then inits devices. The kernel does not use any BIOS functions, but does access the data structure left by the BIOS for system parameters (memory size etc.)

Intel Embedded Module Block Diagram



William N. Eatherton (`weathert@arl.wustl.edu`)
Washington University in St. Louis

Proposed SPC Architecture



Proposed SPC Architecture (Cont.)

- Majority of normal BIOS functions and the NetBSD boot loader code is now irrelevant.
  - No ISA Bus, video, ethernet, serial port, parallel port, hard drive, floppy drive, keyboard, mouse.
  - Very little hardware variables in boot-up (maybe memory size).
  - OS is loaded by network so BIOS and boot strap code do not need to jump through hoops to handle legacy storage media limitations (MBR size, etc.)
- System FPGA provides all necessary system resources
  - The XC4020XL160qfp-2, is the current choice with 20K marketing gates, in a 208 pin QFP package, and a 3.3V core with 5V tolerant pads
  - the PCI slave interface utilizes around%20 of the XC4013 logic resources leaving 8-9K gates for system functions
  - PIT only needs single counter number 0 (other counters in PC are for speaker and ISA DRAM refresh)
  - BIOS is few dozen lines at most (checks memory and occupies the CPU until OS loading is done). The new NetBSD boot loader code will shoulder some of the old BIOS responsibilities.
  - RTC is not really real time but is loaded by network at each reset initialization. The NVRAM in the RTC need not be implemented since normally the legacy BIOS passes all relevant information through the BIOS data area. ***The SPC will not maintain any state between initializations.***
  - PIC only needs to handle 2-3 interrupts (Counter 0, APIC, and maybe one for experimentation FPGA)
  - For all the system resources, they can be designed to emulate the respective legacy device behavior AFTER they have been set up by a legacy BIOS.

William N. Eatherton (weathert@arl.wustl.edu)
Washington University in St. Louis

SPC Details

- Cost per unit (not including development)

Table 1: SPC Cost per Unit

Component	Qty 10
Intel Embedded Module	395
PCB MFG.	150
DRAM (32 MB)	100
System FPGA	78
1 Serial Eprom	15
APIC	150
Optical Interface	300
Misc.	100
Total	1,203

- System FPGA is loaded at power up from a 8 pin serial eprom, also ability to load through an ‘xchecker’ cable gives ability to test new designs before burning EPROMs
- At powerup System FPGA outputs are tristate so all reset lines (active low) will have pulldown resistors.

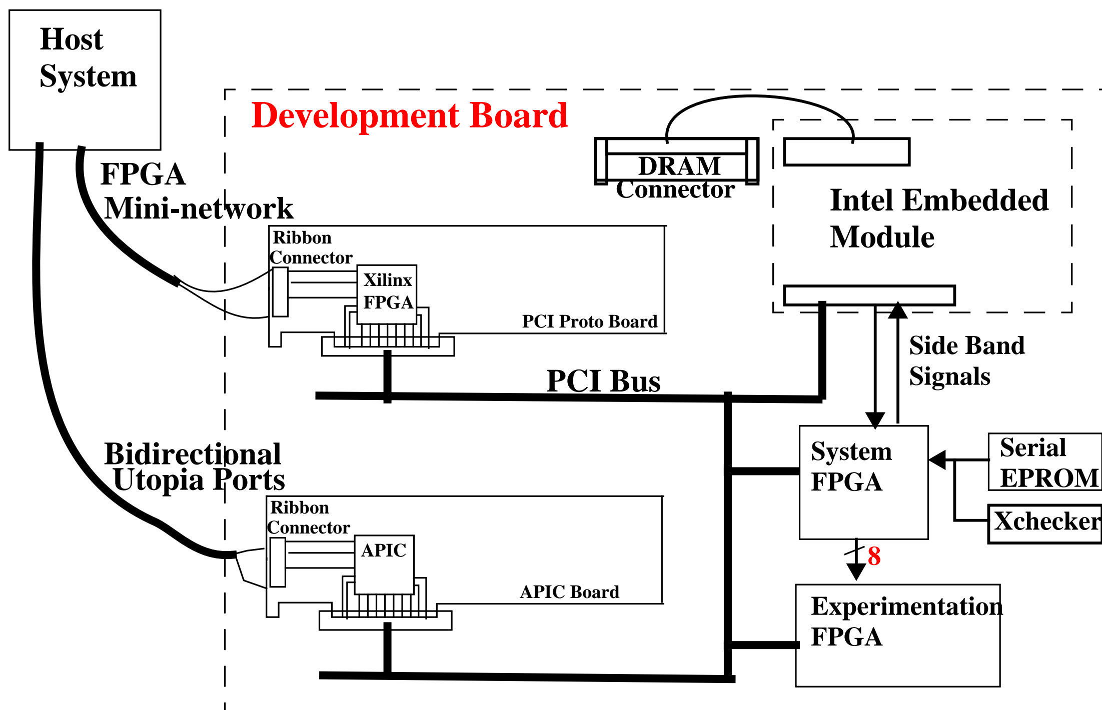
SPC Initialization

1. At power up or after a hard reset, the signal PWROK transitions (via analog reset circuitry) from deasserted to asserted. The PWROK signal enters the System FPGA chip. Initially the System FPGA outputs are all tristated (the reset lines it drives will be pulled down). When the board powers up, the FPGA will download its design from a serial eprom and then “turn on” its pads. When the PWROK signal is detected as asserted the system FPGA will with appropriate timing deassert the CPU and PCI reset signals.
2. Pentium does internal initialization and then does a memory read from address 0xfffffff0. At reset, the North Bridge chip has this memory area marked as disabled so read is passed to PCI bus.
3. The system FPGA will have this address hardwired as a decode to its embedded BIOS ROM. The first word of the bios is read out onto the PCI bus and passed to the CPU.
4. The CPU executes this instruction, which is usually a jump to the top half of the normal BIOS memory space 0xffff0000 (since we are 16 bytes from the end of memory).
5. The only POST check the BIOS will do is a memory check, where each physical memory address is written and read. When done, the results of the test will be written into a register of the FPGA.
6. The BIOS code at this point has the CPU do a simple 1000 iteration loop within an infinite loop that exits when a register in the System FPGA equals a non zero value(which is inited to zero at power up).
7. At some point the appropriate control processor (CP) which knows the control VCI of the APIC on the SPC under discussion, will start sending said APIC control cells.
8. First the CP processor has the APIC fetch the memory POST test results from the FPGA and send it back. If the test failed the CP processor halts OS loading.
9. Next the CP sets up the global APIC registers via control cells.
10. Then the CP has the APIC write a Rx descriptor starting a location 0x00000000. It describes an AAL5 packet splitting receive buffer 64Kbyes long at some set location in memory. Then descriptors are written for additional 64K buffers that will adequately hold the kernel. Finally, the CP resumes the connection and the APIC fetches the first descriptor

11. The CP then sends the first 64KB of the NetBSD OS as an AAL5 frame, when the RX buffer is full the APIC fetches the next descriptor until the entire OS (new boot loader code, kernel, and application) has been loaded into one contiguous block of memory. The remote control processor then reads all the returned descriptors and checks that the CRC values are correct.
12. At this point the DONE register in the system FPGA will be set to a non-zero value, and the next time the CPU checks the register it will jump out of its loop to the beginning of the OS code which will be the “newbootloader”.
13.. the “newbootloader” does several BIOS emulation tasks: it writes a few parameters to the BIOS data area (Memory size, the time and date from the RTC), inits the interrupt table, enables the APIC and writes its base address for slave access.
14. One the BIOS emulation tasks of the “newbootloader” is done, it executes the kernel (passing in the boot flags [i.e. single user boot, etc.]).
15. the kernel inits memory management, then inits devices. The kernel does not use any BIOS functions, but does access the BIOS data area left by the emulated bios.

SPC Development

- Testing of APIC with development board
  - * Host PC and development board are connected by both a pair of APIC cards for loading of OS, and by the FPGA mini-network for probing of the development system.
  - * Logically, the development board with APIC card will look exactly like the final SPC design. A gigabit switch line card and the APIC card could be interconnected to duplicate the final SPC functionality for testing.



William N. Eatherton (weathert@arl.wustl.edu)
Washington University in St. Louis

Embedded NetBSD

- New BIOS has been hand coded (under 32 bytes) assembled and tested in PC
  - * Functionality is to put CPU into a loop, CPU counts for a while, then the CPU polls a memory location to see if it has been set to done by remote processor
  - * While CPU loops, OS can be loaded into main memory through APIC by remote control
- RAM File System has been embedded into NetBSD Kernel
- Bootloader modifications currently being worked on
  - * Configure APIC
  - * Look at boot strap pins to determine amount of main memory
  - * Initalize BIOS data and interrupt vector areas (Bios isn't doing it)
  - * Jump to beginning of Operating System code