PCI Coprocessor Expansion Card

Project Proposal

Alex Barabanov
Shawn Crabb
Tom Gongaware
David Palchak

April 27, 2004

Motivation

Currently, the majority of personal computers are used repeatedly to perform a very small daily subset of possible tasks. The average PC user runs roughly a dozen or so applications that account for the vast majority of usage. The most common of tasks tends to be word processing, web browsing, reading email, storing and listening to digital media, etc. Inherent in these everyday tasks is a shared set of computations that must be performed frequently. One example of such a computation is encryption. Every time a user shops online, checks their email through a secure server, or runs some kind of automatic software update tool, the same algorithm is executed to encrypt and decrypt the data being transmitted. If such an algorithm could be off-loaded to dedicated hardware, then the CPU could concentrate on floating point calculations and other advanced tasks.

Introduction

Our proposed project is to design and construct a PCI expansion card containing an embedded microprocessor and associated memory. This device will feature EEPROM for storing encryption algorithms and any required hardware management routines. A bank of RAM will also be integrated on the card to provide buffer and scratch space. The device capabilities will be accessed through a kernel-space driver that is coupled with one or more user-space shared libraries. The purpose of the libraries would be to replace current routines with a drop-in solution that could take advantage of the processing power added by the card. The proceeding sections describe each of these design aspects in greater depth, followed by a tentative task layout and bill of materials.

Hardware Design

The principle hardware for our project consists of a PCI development platform and the processor that the platform will host. For the PCI platform, we have selected the 9030 Rapid Development Kit offering from PLX Technologies. This kit provides a PCI 2.2 compliant development board with an integrated PLX 9030 target interface chip. The 9030 device provides a 32-bit local bus interface capable of operating at up to 60 MHz. The RDK also includes a local bus memory controller with 16KB of SRAM, a large number of surface-mount prototype areas, debugging headers, and sample device drivers.

For the processor we are currently targeting the Intel 960 family. This line of devices offers a 32-bit MIPS platform with many high performance options. We have identified the HA part, operating at 3.3V with a core speed of 33Mhz, as the most fitting for our needs. We will be purchasing the device in a 208-pin SQFP package, which will fit into the largest SMT prototype area on the RDK.

Auxiliary hardware required for our design includes a 4MB flash ROM, additional RAM, and a 33MHz fixed oscillator for the local bus. The flash ROM is a serial ROM in an 8-pin dip package, selected to match the socket provided on the development board. The RAM will be two chips of 4MB dual-port Cypress SRAM. These chips will be inserted in place of the current SRAM parts included with the RDK. Lastly, the oscillator will replace the 60 Mhz Ecliptek part shipped with the RDK. The new part will be ordered directly from Ecliptek, with single quantity pricing and availability already confirmed.

Software Design

The software required to successfully operate the described hardware will be comprised of three components, a kernel-mode device driver, two or more user-mode system libraries, and a set of embedded software routines for processor management and task execution. We have decided to target a GNU/Linux 2.4 system as our initial host platform for stability and documentation reasons. Additionally, several compilers for the i960 are available for free that run under Linux, including GNU gcc and Intel's CTOOLS.

The device driver will be responsible for all direct interaction with the 9030 target device. This code will be a kernel module based on the Linux reference drivers provided as part of the RDK. The driver will provide two sets of interfaces to user-mode applications. The primary interface will have the necessary methods for storing and retrieving data from the card's memory and signaling the processor for execution requests. The second interface will allow for an application to control a run-time reprogramming of the flash ROM. This interface will be used to store the embedded software to the card.

At this juncture, we have identified libcrypt.so and libssl.so as the two system libraries we intend to modify to use our hardware. These libraries will be augmented with the necessary code to export the work of certain tasks, such as MD5 checksumming and RSA encryption/decryption, to the card's processor. The decision to use standard system libraries

was made because it will allow applications to use the hardware without needing to be recompiled.

Complementing the modified libraries will be many embedded subroutines that perform the tasks originally performed in the libraries. The majority of this component will be hand coded in assembly, because this is potentially the most performance critical aspect. It is our projection that offloading computations to the card will boost the average throughput of the host system when under very high CPU loads.

Interfaces

In designing our project, we have identified three principle interfaces: the processor to PCI interface, the embedded software to processor interface, and the PCI device driver to user-mode API.

Processor Hardware Interface

The hardware interface between the processor and the PCI controller presents several critical issues. The first involves synchronization and message passing between the PLX 9030 chip and the processor. The 9030 RDK provides a 32-bit local bus managed by the 9030 controller for attachment of on-card devices. However, the local address bus pins of the 9030 part are output only, which means that the i960 could not be simultaneously connected to the bus. The solution for this is the on board SRAM is configured such that the second port is tied to a second local bus. It is on this second bus that the processor will reside, allowing both the processor and the 9030 controller access to the contents of RAM. Thus, communication between the two devices will be accomplished using a shared-memory technique such as queued mailboxes and dedicated interrupts. The exact implementation of a synchronization scheme will depend almost entirely on the capabilities of the 9030 part.

A second major consideration is the segmentation of the memory space between the EEPROM and the RAM. The processor will likely not have exclusive access to the EEPROM, as the PCI controller may need some custom initialization code. The solution which we are currently looking at implementing for this problem will involve augmenting the SRAM with an EEPROM and simply extending the available address space. This would also make the EEPROM DMA addressable over the PCI bus, which could be taken advantage of by the flash update components.

Processor Firmware API

The API for the code executed by the processor will need to be defined in advance to ensure compatibility. This will primarily consist of defining the EEPROM/RAM address map and the memory image assigned to each thread. Other areas that will require attention include the sequencing of operations for a flash update, exceptional event handling, power-on initialization sequencing, and thread dispatch and suspension procedures. Most of these design issues are currently pending the acquisition of complete processor and device driver documentation.

PCI Device Driver

The interface between the PCI controller and the Linux kernel is going to be a formidable task. To our advantage, though, the RDK provides a Linux reference driver with source code that we intend to extend for our project. Additionally, a large amount of documentation on the Linux 2.4 driver interface is freely available online. These resources will help reduce the development time for this component.

The device driver will implement a common subset of functions that can be adapted to initiate and control a wide variety of PCI operations. These include loading and fetching data to and from the card‘s memory, initiating, pausing, and terminating execution of an embedded thread, reading the processor status, and writing to the EEPROM. This type of design will allow for the card to used to perform different specialized tasks, such as compression or software RAID computations, without the need for any driver modifications.

The device driver will also be responsible for ensuring that tasks exported to the card do not overload its processing capabilities. Currently, our idea for accomplishing this is to instantiate a fixed number of threads running on the embedded processor and structure the driver to only grant resource access to that many clients. This could be accomplished through an object-oriented interface where the number of threads is the maximum number of “resource handles” the driver will instantiated at a given time.

Testing and Integration

Hardware

The primary focus of the hardware integration will be configuring the RDK and 9030 controller to coexist with the i960. This interaction will be tested once a beta device driver has successfully written to and read from the memory on the card through the 9030 controller.

The test will consist of loading some data to memory and then using the 9030 to signal a processor interrupt. The interrupt handler code for the processor will perform some simple task on the data, which can then be read back and validated. This will also test the processor's access to the EEPROM containing the assembly code.

Most of the interaction between the various hardware components are very interdependent and difficult to isolate for testing. For this reason, most of the hardware testing will occur at once. Furthermore, troubleshooting will require extensive use of digital scoping of bus signals and the status monitoring tools provided as part of the RDK.

Software

The software has two main levels of integration, software-to-software and software-to-hardware. One part of implementing the software will be to divide the libcrypt and libssl routines in Linux. The common routines in these libraries will have to be able to successfully send data out and then react when they receive data back. This will be the main focus of testing for these routines. For this, completion of the hardware is unnecessary, as a dummy interface will be generated to simulate hardware behavior.

In order to test the firmware and its interactions with the hardware, we will implement a simple "hello world" test program. This test is similar to the integration test briefly mentioned in the hardware sections. The purpose of the "hello world" program will be to verify that the OS is recognizing the hardware and its functionality. Successful execution of this test program is one of the prominent milestones of our project.

The final testing phase of the software will be verification that it all works together as expected. Expected operation includes the encryption/decryption calls by the OS being off-loaded onto the co-processor via PCI. Once this is confirmed we'll move to testing the receiving of the encrypted/decrypted information back to the main processor. Finally, we'll test the ability of the hardware to write updates to the EEPROM by attempting such a task.

Risks

Hardware

The greatest risk with regards to the hardware is fitting and soldering the surface mount parts to the RDK. Our analysis of the prototyping areas of the RDK conclude that suitable target areas exist for the extra hardware our project requires. However, none of us have experience with large scale SMT parts, so this will require more time than otherwise necessary.

To account for this time demand, we have already begun to order the RDK and processor. Our plan is to complete the SMT placement and soldering over the summer using the equipment available in Al Davis' lab.

Software:

The only significant risk with the proposed software stems from our inexperience with the chosen target processor and with Linux device drivers. The former we do not believe to be a serious hindrance, simply because we have similar experience with other embedded targets, including MIPS processors in general. The latter, though, will require thorough reading of kernel documentation and source before next fall so coding work can begin immediately. Testing of the device driver will be difficult until the hardware is at least partially assembled, so this area of integration will certainly need to be allotted a sufficient portion of project time

Tasking

Hardware:

Before Fall Semester:

- **Processor documentation acquisition:** obtain printed copies of developer's documentation and/or reference guides
- **PCI development kit documentation acquisition:** obtain printed copies of developer's documentation and/or reference guides
- **Auxiliary Hardware Identification:** identify additional hardware needed for RDK/processor interaction
- **Device acquisition**: purchase selected processor, development kit, auxiliary hardware, EEPROM, additional RAM
- **Device packaging accommodation:** solder SMT parts onto development board or wire-wrap carrier boards

Starting in Fall Semester:

- **Auxiliary processor circuitry:** construct external circuitry required to begin experimentation with (1.5 weeks)
- **"Hello World" program execution**: develop and execute a simple assembly program to verify microprocessor operation (1.5 weeks)
- **External memory interfacing**: interface microprocessor with RAM/EEPROM segments (1.5 weeks)
- **External memory interface testing**: rigorously test microprocessor/memory interaction (0.5 weeks)
- **I/O controller initialization:** develop an initialization routine for PCI controller
- **I/O controller interfacing**: interface microprocessor with PCI controller; establish DMA protocol (1.5 weeks)
- **I/O controller interface testing**: rigorously test microprocessor/PCI controller interaction (0.5 weeks)

- **Firmware Programming model documentation**: document available resources, calling conventions, and system requirements for firmware routines according to hardware design (1 week)
- **PCI DMA synchronization testing:** test PCI DMA synchronization in live system (1 week)
- **Integration Testing:** test processor/firmware interaction, PCI bus response, operating system response, total system operation (all remaining time)

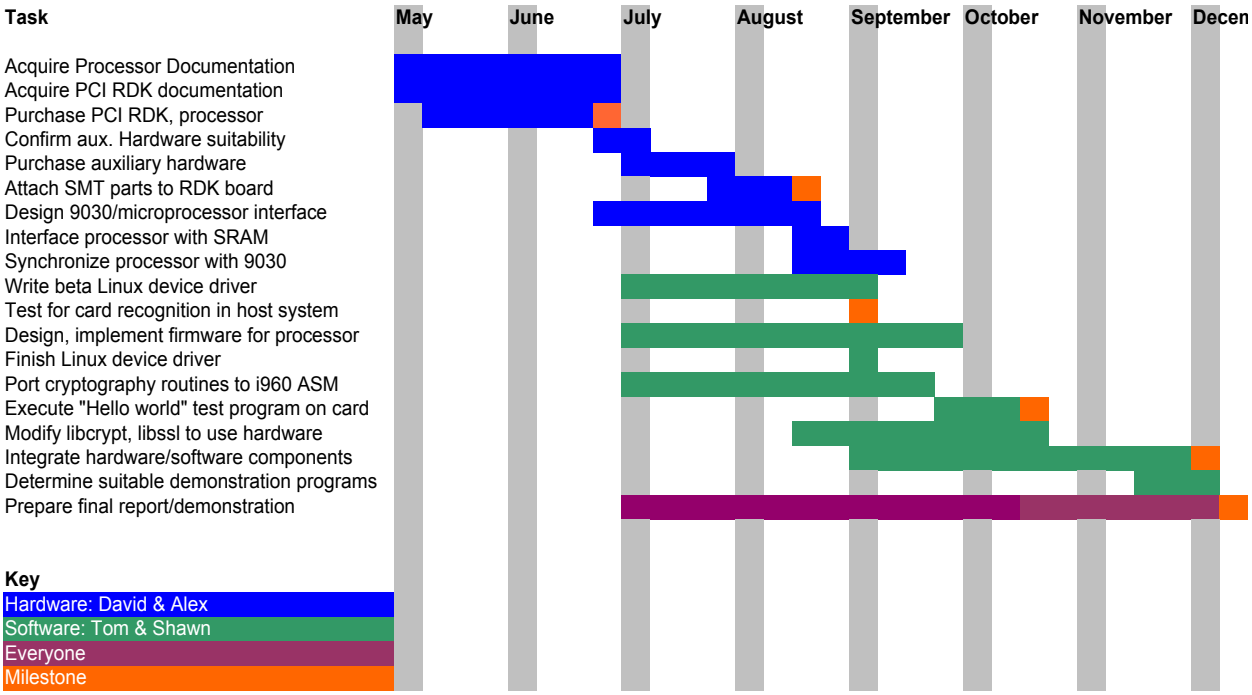
Software

Before Fall Semester:

- **Obtain API documentation:** Acquire documentation for Linux kernel, device drivers, 9030 PCI target device, libcrypt, libssl, and i960 processor family
- **Plan functional hierarchy:** Create flow charts of what needs to happen in the firmware and software.

Starting in Fall Semester:

- **Write device driver:** extend RDK reference driver to provide necessary interface routines for hardware access (6 weeks)
- **Write firmware:** write firmware code to manage processor state (1 week)
- **Write assembly cryptography routines:** write embedded code to execute cryptographic functions performed by system libraries (2 weeks)
- **Simulate Assembly code:** thoroughly simulate firmware and crypto routines (1 week)
- **Modify system libraries:** modify system libraries to export work to hardware (2 weeks)
- **Integration Testing:** test processor/firmware interaction, PCI bus response, operating system response, total system operation (all remaining time)



Bill of Materials

Intel FC80960HA33 processor Mouser Electronics Arrow Electronics	1	$90 Single unit pricing confirmed
PLX 9030 Rapid Development Kit PLX Yahoo Store Lang Sales (Littleton, CO)	1	$350.00 Single unit pricing confirmed
Ecliptek 33 Mhz Programmable Oscillator (EP1345PD-33.000M) Ecliptek Mouser Electronics	1	$18.20 Single unit pricing confirmed Lead time: 2-3 days
Atmel AT29C010A-12JC Flash ROM (4 MB) Arrow Electronics Mouser Electronics	1	$7
Cypress CY7C019-12AC Dual Port SRAM (4 MB) Compass (SLC, UT) Advanced Technical Sales (Englewood, CO)	2	$58 Pricing confirmed
Discrete Parts (R's and C's) Mouser Electronics Arrow Electronics	TBD	<$10