PCI Coprocessor Expansion Card

Alex Barabanov
Shawn Crabb
Tom Gongaware
David Palchak

Computer Engineering
Senior Design Project
Fall 2004

Table of Contents

MOTIVATION: 3
INTRODUCTION: 3
USER INTERFACE: 3
  HARDWARE: 3
  SOFTWARE: 5
DESIGN 6
  HARDWARE: 6
  SOFTWARE: 7
DESIGN DECISIONS 8
CONCLUSIONS 10
ACKNOWLEGMENTS 11
APPENDIX A (session log): 12
APPENDIX B (source code): 16
  FIRMWARE: 16
    bit.h 16
    bus.h 16
    bus.c 17
    main.h 21
    main.c 21
    md5.h 25
    md5c.c 26
    types.h 32
  SOFTWARE: 33
    Makefile 33
    manager.c 34
    md5test.c 37
    plxinterface.h 39
    plxinterface.c 40
    Testfile.c 41
    Teststrings.c 43

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:

The purpose of this project was to develop a proof-of-concept PCI expansion card containing an embedded processor and associated memory. The function of this card was to execute a single encryption or compression routine very efficiently, so that the host system could offload work to the card. The actual offloading procedure was originally intended to be wholly invisible to user applications, so that transparent hardware utilization could occur. Time constraints did not permit this level of system integration, but application level hardware utilization was achieved. Overall, this project succeeded in demonstrating the potential for increased computing performance through modular hardware, the core concept initially presented.

USER INTERFACE:

HARDWARE:

As is common with many other types of PCI expansion cards, the user interface for our project is extremely minimal. One of our design goals was to allow applications on the host system to transparently take advantage of the additional hardware capability. Though we did not accomplish this transparency to the extent planned, we still managed to require very little user effort to install and operate the hardware.

Installation of the hardware consists of two parts, physically inserting the card and loading the kernel device driver. The card is designed to fit into a standard 32-bit PCI expansion slot on the motherboard of a computer. The expansion slot can be either 3.3V or 5V and feature either a 33 Mhz or 66 Mhz bus speed.

Once the card has been physically added to a system, a Linux host operating system is needed to install the kernel device driver. The driver is written to operate with 2.4 series kernel. Our test system ran kernel version 2.4.20, the stock RedHat Linux 9 build. The driver is obtained by compiling a set of source files provided by PLX as part of the development kit we purchased. A well documented makefile is provided along with the source to aid in the compilation of the driver.

Installation of the compiled kernel driver is accomplished by executing a shell script installed along with the driver sources. The shell script performs two primary actions. It first attempts to load the kernel object containing the driver code into the kernel. If this action succeeds, it indicates that the driver was successfully able to find a PLX interface controller on the PCI bus. The second action performed by the installation script is creation of the necessary filesystem device entries that allow user applications to communicate with the driver. These entries are created under the /dev/plx/ subdirectory with a major node ID of 254. For our project, these device entries had names of the form /dev/plx/Plx9030-X, where X is a integer between zero and four.

Accessing the hardware from a user application involves calling several initialization functions supplied by a general purpose interface library that accompanies the kernel driver. The calls used by our software perform the following tasks, in order: locate the PCI address of the PLX device, open the PLX device and initialize its registers, and map a region of the calling process' address space to the SRAM on the card. The last step returns to the application an address that corresponds to the first addressable byte of the memory on the card. This value is then assigned to a pointer, and the pointer can be used to directly read and write to the card's RAM.

Included in Appendix A is a session log showing the correct steps for installing and compiling the sources for the kernel driver and interface library. Appendix B contains a listing of the source for a simple test application that accesses the hardware and performs several reads and writes.

SOFTWARE:

Our original software design was to have no user interface. Once installed, as described in the hardware user interface section, the entire system was designed to be invisible to the user. However, for the sake of a convincing demonstration, and for development, we created a several test applications for users to directly interface with this system.

There are two different types of test applications available to the user. The `md5test` code is a standalone program that tests for correctness of the MD5 algorithm firmware implementation. It takes a filename as a command line argument and processes this file on the card in chunks. It also digests the file using the builtin system libraries. Upon completion of both tasks, the program compares the card's digest to the system digest and notifies the user of the success or failure.

The other test applications, `Testfile` and `Teststrings`, use a TCP/IP socket to connect to a third application called `manager`. The `manager` program features a standard Unix daemon-style design and is intended to run as a background process. The `manager` program starts by binding to a listening socket and waiting for client connections. When a client connects, the program reads data to be digested, writes the data to the card, waits for the card to perform the digestion, reads in the result, sends it back to client, and closes the socket. The process then loops back and continues listening for new client connections.

Two applications were written to work in tandem with the `manager` program. `Teststrings` prompts the user for strings to digest, and when one is entered, it sends it opens a socket to the `manager` program and executes a transaction. `Testfile` takes a filename as a command line argument. The program splits the file into smaller chunks and digests each of these using the same semantics as `Teststrings.`

As mentioned, the intended interface was to be entirely transparent to both the user and to user applications. This was to be accomplished through the use of a background process operationg exactly like `manager.` Coupled with this was to be a modified crypto system library containting socket code similar to that in `Teststrings`. User applications could continue to use the standard system library calls, never knowing that behind the scenes the actual code execution was occurring on the card. Unfortunately, due to time constraints, such an implementation was never fully accomplished.

DESIGN

HARDWARE:

The hardware is designed around two separate busses that each connect to an independent port on the SRAM. The reason for the two bus design is due to a limitation of the PLX 9030 PCI target interface chip. The address pins on the device are designed for output only, meaning that the 9030 is incabable of ever reqlinquishing control of the address lines.

The 9030 device sits on the left bus, referring to the left memory port. This bus has a 32-bit width running at 33 Mhz and has address lines configured for 32-bit word addressing. The left SRAM port is configured for burst-mode accesses, which the PLX performs with the assistance of a small 20-pin GAL device (Generic array logic). The GAL uses the read/write, address strobe, burst last, and chip enable signals from the 9030 to generate read/write, address strobe, and chip enable signals for the SRAM.

The development kit ships with the right memory port disabled using appropriate pull-up and pull-down resistors. All of the data, address, and control lines for the right memory port are connected to a 0.1" standard post header to facilitate additional interfacing. For our project, this bus was connected to an Atmel ATmega128L microcontroller running at 8 MHz. Due to pin count restrictions on the Atmel part, the bus was narrowed to 16-bits by connecting the upper 16 data lines to the lower 16 data lines. Bus contention was then controlled through the lower and upper byte enable control lines for each physical SRAM chip. Altogether, the right bus required 16 I/O pins for data, 13 output pins for addressing, and 6 output pins for control signals. Table 1 shows the bus pin assignments for the Atmel part. Additional control lines not shown in the table were hardwired using pullup and pulldown resistors to appropriate values. For the right side bus none of the burst capabilities of the SRAM were used nor was the internal address counter.

Pin Reference	***Bus Component***
Port A	Address[0-7]
Port B [0-5]	Address[8-13]

Pin Reference	***Bus Component***
Port C	Data[0-7], Data[16-23]
Port D	Data[8-15], Data[24-31]
Port E[0-5]	[0] Address strobe [1] Lower word enable [2] Upper word enable [3] Read/Write select [4] Output Enable [5] Chip Enable

Table 1: Atmel Port Configuration

SOFTWARE:

Our software design had a number of layers, each adding a level of complexity. We started out with simple, direct communication with the card, specifically for testing. Then, we built in functionality into the `manager` program to receive data through a socket (from anywhere) and send it to the card. Finally, we built a simple communication protocol into the `manager` program to accept data over the socket in a certain way. This resulted in the `KernelMan` program. Then, we aimed to build functionality into the kernel libraries to send the data, via our designated protocol over a socket connected to `KernelMan`, which would then perform the task of sending the data received from the kernel libraries to the card.

The functionality of the `KernelMan` program is nearly identical to that of the `manager` program. When the MD5 calls are made in a user application (`MD5_init`, `MD5_Update`, and `MD5_final`), our design was to forward this request, through `KernelMan`, to the card to compute. `KernelMan` was to then wait (polling the card) until the digest was returned from the card and forward it on to the process. Due to time constraints, we were unable to make the necessary changes in the kernel library functions to send the data over the socket, so the communication protocol of `KernelMan` was not needed. Given more time, `KernelMan` would have been our main program that users would run on their system to manage the various calls to the MD5 functions.

Since `KernelMan` was never fully functional, the main program is `manager`. The `manager` code is run from the command line or (optionally) from a system startup script. The `manager` listens on port 8200 as a stream socket for incoming messages. It expects the messages to be sent in the following format:

Message 1: Send the size (4-byte integer) of the message to be sent.

Message 2: Start sending the message.

The buffering of the message takes place in the kernel memory because of the sockets, so there is no need to worry about receiving the entire message into a large buffer. When data is available, the `manager` transfers it directly from the socket buffer onto the card. Currently, `manager` polls the flag field of memory to know when the card has completed computing the digest. Upon completion, `manager` sends back the digest consisting of four 32-bit integers to the requesting application.

Our simple communication protocol was designed around the use of the MD5 algorithm. The MD5 algorithm makes use of 3 functions, MD5_Init(), MD5_Update(), and MD5_Final(). To each of these we assigned an integer designation (0, 1, and 2, respectively). The first byte of a message send by a client to the `manager` was the desired function to execute. The MD5_Init() function (command 0) performs some initialization, so no further data or arguments need to be sent to the card. The MD5_Update() function needs data to send to the card, so a proper message consisted of a single byte specifying the function, four bytes specifying the size of the data in bytes, and then the data bytes. The MD5_Final() function completes the MD5 algorithm, performing any necessary padding. When KernalMan gets executes this command, it sends the message digest back to the calling function when it is finished.

DESIGN DECISIONS

The decision to select the PLX 9030 PCI controller chip and associated development kit was made based on the perceived design time savings of the development kit. The kit included a number of the critical hardware components we needed, including a prefabricated PCI card, an interface chip, integrated onboard SRAM, and solder footprints for a variety of surface mount components. The development kit also included a critical software component, the Linux kernel device driver. Each of these features turned out to play a significant part in the eventual success of the project.

As mentioned previously, the development kit that we purchases shipped with a single SRAM installed and configured for 16-bit access. To increase the potential performance of our hardware, we chose to augment the card with an additional SRAM, to bring the total memory size up to 8192 bytes (8 KB). The development kit was already designed to accommodate a second SRAM, and installation simply required shifting a set of resister networks so that the left bus would use 32-bit word addressing.

The project design set forth last spring included an Intel 960 HA 32-bit microprocessor in place of the Atmel part. This particular Intel device was strictly a microprocessor, and it did not feature any additional interface logic nor, most importantly, any general purpose I/O. This was initially perceived to be acceptable, as we felt confident that given the simplicity of the SRAM interface that the 960 processor could be configured to work correctly. However, as the development progressed, other hardware problems demanded significantly more time than originally anticipated. With just under 6 weeks remaining in the semester, full blown integration of the 960 with the SRAM had not even begun. Coupled with this looming task was the inexperience our team had with the scarce software development tools available for the part. Weighing out the options with a careful eye on our slipping schedule, the decision was made to switch processors.

The Atmel part was chosen for two primary reasons. First, it possessed the general purpose I/O capabilities that the Intel part lacked. This was important because it allowed us to retain explicit control over all aspects of the interaction with SRAM. The second supporting factor for the selection was the development experience with this part that was share by two members of the team. This experience translated into an incredible savings in development time, allowing attention to be focused on other persisting hardware problems.

In creating our manager process, one major decision we made was to use sockets for communication between processes. This solved a number of communication problems. By using sockets, we are able to limit the number of processes trying to access the card at one time. Because our system was meant to enhance performance, it seems natural to not let requests for use of the card stack up in any sort of queue. So, if the card was already busy, it would just compute the MD5 digest on the host processor. This was simple with sockets, because we just defined them not to allow more than one connection at a time, so if a connection fails on the library side for any reason, the library then just finishes the execution internally. Another simplification available to us through the use of sockets was that the kernel provided nearly limitless buffering, which reduced the complexity of our code.

Our original design included the implementation of the AES encryption algorithm (Rijndael). After looking into reference implementations of this algorithm, however, we decided to implement the MD5 check sum algorithm. This was largely due to the fact that the fast way to implement Rinjndael algorithm is to use pre-computed tables. These tables could not be used on our card, because they were larger than the memory available to us for storage. While it is possible to compute the values in the tables on the fly, the loss of performance on our 8-bit processor would have been excessive. The MD5 alorithm, on the other hand, proved to have a very straightforward implementation as well an interface that was easily ported our system structure.

CONCLUSIONS

Overall, we feel we accomplished our main goal of providing a proof-of-concept PCI-based co-processor. There were a number of things we could not fully implement, largely due to time constraints, but we still feel as though the project was a success. There are two main things that we feel would have brought the end results of this project a lot closer to our original concept. First, we originally planned to have the whole system working transparently, so the process that called the kernel libraries wouldn't know their data was processed on the card. This added functionality would have really shown that our fully integrated system was a viable option. Second, another of our original concepts was to increase performance. There were a number of decisions we made that made this kind of performance increase impossible. For example, choosing a slower processor for the card, while making development more realistic, slows the overall performance of the off-loaded process. Another performance hampering decision we made was to use polling, or busy waiting, to communicate between the host and the Atmel part. Implementing an interrupt-driven system would have been much more efficient, but also more difficult.

Overall we had to make some trade-off decisions, choosing options that were easier or faster to implement to ensure a final working product by the demonstration deadline. This strategy worked, showing that even though we were unable to implement it completely within the time limits of a school semester, it is possible to create the system that we originally conceptualized.

ACKNOWLEGMENTS

We would like to thank the following people for their contributions to and assistance with this project.

Control4 Comporation, for their loan of a JTAG programmer and for providing access to their SMT rework equipment.

Mark Burnett, of Thorson Rocky Mountain, the local Atmel sales representative, for obtaining for us free of charge two Atmel Atmega128L microcontroller samples.

Al Davis, our project advisor, for access to his research laboratory and soldering facilities as well as his generous loan of expensive and helpful test equipment.

APPENDIX A (session log):

```
Script started on Sun 12 Dec 2004 07:25:04 PM MST

[pci@PrimeImplicant pci]$ mount /dev/cdrom
[pci@PrimeImplicant pci]$ cd /mnt/cdrom
[pci@PrimeImplicant SdkFiles]$ cd PciSdk/
[pci@PrimeImplicant PciSdk]$ cd Linux_Host/
[pci@PrimeImplicant Linux_Host]$ cp PlxLinux.tar ~/
[pci@PrimeImplicant Linux_Host]$ cd
[pci@PrimeImplicant pci]$ tar -xf PlxLinux.tar
[pci@PrimeImplicant pci]$ cd PlxLinux
[pci@PrimeImplicant PlxLinux]$ pwd
/home/pci/PlxLinux
[pci@PrimeImplicant PlxLinux]$ export PLX_SDK_DIR="/home/pci/PlxLinux"
[pci@PrimeImplicant PlxLinux]$ make
for i in linux/api linux/PciDrvApi linux/samples/ApiTest
linux/samples/DSlave linux/samples/DSlave
linux/samples/DSlave_VirtualAddr linux/samples/InterruptEvent
linux/samples/PlxCm linux/samples/Sgl; do make -C $i all; done
make[1]: Entering directory `/home/pci/PlxLinux/linux/api'
Building: PlxApi
Compiling: PciApi.c
Creating archive: PlxApi.a...
Library file "Library/PlxApi.a" built successfully.
make[1]: Leaving directory `/home/pci/PlxLinux/linux/api'
make[1]: Entering directory `/home/pci/PlxLinux/linux/PciDrvApi'

Building: PciDrvApi
Compiling: PciDrvApi.c
Creating archive: PciDrvApi.a...
Library file "Library/PciDrvApi.a" built successfully.

make[1]: Leaving directory `/home/pci/PlxLinux/linux/PciDrvApi'
make[1]: Entering directory `/home/pci/PlxLinux/linux/samples/ApiTest'
Building: ApiTest
Compiling: ApiTest.c
Compiling: ConsFunc.c
Compiling: PlxInit.c
Compiling: RegDefs.c
Compiling: Test9030.c
Compiling: Test9050.c
Compiling: Test9054.c
Compiling: Test9656.c
Compiling: Test9080.c
Linking application: ApiTest
Executable file "App/ApiTest" built successfully

make[1]: Leaving directory `/home/pci/PlxLinux/linux/samples/ApiTest'
make[1]: Entering directory `/home/pci/PlxLinux/linux/samples/DSlave'

Building: DSlave
Compiling: DSlave.c
Compiling: ConsFunc.c
```

```
Compiling: PlxInit.c
Linking application: DSlave
Executable file "App/DSlave" built successfully

make[1]: Leaving directory `/home/pci/PlxLinux/linux/samples/DSlave'
make[1]: Entering directory `/home/pci/PlxLinux/linux/samples/DSlave'

Building: DSlave
Linking application: DSlave
Executable file "App/DSlave" built successfully


make[1]: Leaving directory `/home/pci/PlxLinux/linux/samples/DSlave'
make[1]: Entering directory
`/home/pci/PlxLinux/linux/samples/DSlave_VirtualAddr'

Building: DSlaveVa
Compiling: DSlave_VirtualAddr.c
Compiling: ConsFunc.c
Compiling: PlxInit.c
Linking application: DSlaveVa
Executable file "App/DSlaveVa" built successfully

make[1]: Leaving directory
`/home/pci/PlxLinux/linux/samples/DSlave_VirtualAddr'
make[1]: Entering directory
`/home/pci/PlxLinux/linux/samples/InterruptEvent'

Building: InterruptEvent
Compiling: InterruptEvent.c
Compiling: ConsFunc.c
Compiling: PlxInit.c
Linking application: InterruptEvent
Executable file "App/InterruptEvent" built successfully

make[1]: Leaving directory
`/home/pci/PlxLinux/linux/samples/InterruptEvent'
make[1]: Entering directory `/home/pci/PlxLinux/linux/samples/PlxCm'

Building: PlxCm
Compiling: ConsFunc.c
Compiling: CommandLine.c
Compiling: Monitor.c
Compiling: MonitorCommands.c
Compiling: PciDevice.c
Compiling: RegisterDefs.c
Linking application: PlxCm
Executable file "App/PlxCm" built successfully

make[1]: Leaving directory `/home/pci/PlxLinux/linux/samples/PlxCm'
make[1]: Entering directory `/home/pci/PlxLinux/linux/samples/Sgl'

Building: Sgl
Compiling: Sgl.c
```

```
Compiling: ConsFunc.c
Compiling: PlxInit.c
Linking application: Sgl
Executable file "App/Sgl" built successfully

make[1]: Leaving directory `/home/pci/PlxLinux/linux/samples/Sgl'

[pci@PrimeImplicant PlxLinux]$ su
Password:
[root@PrimeImplicant PlxLinux]# cd linux
[root@PrimeImplicant linux]# cd driver
[root@PrimeImplicant driver]# ./builddriver 9030

Building: Pci9030
Compiling: Pci9030/ApiFunctions.c
Compiling: CommonApi.c
Compiling: Dispatch.c
Compiling: Driver.c
Compiling: EepromSupport.c
Compiling: ModuleVersion.c
Compiling: Pci9030/PlxChip.c
Compiling: Pci9030/PlxInterrupt.c
Compiling: PciSupport.c
Compiling: SupportFunc.c
Linking driver: Pci9030.o
Device driver "Pci9030/Driver/Pci9030.o" built sucessfully.


[root@PrimeImplicant driver]# cd ..
[root@PrimeImplicant linux]# cd PciDrvApi/
[root@PrimeImplicant PciDrvApi]# cd Library/
[root@PrimeImplicant Library]# cp PciDrvApi.a /home/pci/tools/lib
[root@PrimeImplicant Library]# cd ../..
[root@PrimeImplicant linux]# cd api/Library/
[root@PrimeImplicant Library]# cp PlxApi.a /home/pci/tools/lib

[root@PrimeImplicant Library]# cd ../../../bin
[root@PrimeImplicant bin]# ./modload 9030

   ****************************************************************
   * NOTES:                                                       *
   *                                                              *
   *  You must be superuser, logged in as root, or have sufficient *
   *  rights to install modules or this script will not work.      *
   *                                                              *
   *  A warning regarding 'kernel tainting' is normal.  This is    *
   *  because the PLX driver is marked with a 'Proprietary'        *
   *  license tag, not GPL.  For more information, please refer    *
   *  to:                                                         *
   *        http://www.tux.org/lkml/#export-tainted                *
   ****************************************************************

Installing module (Pci9030)....
insmod version 2.4.22
```

```
Using /home/pci/PlxLinux/linux/driver/Pci9030/Driver/Pci9030.o
Symbol version prefix ''
Warning: loading
/home/pci/PlxLinux/linux/driver/Pci9030/Driver/Pci9030.o will taint the
kernel: non-GPL license - Proprietary
  See http://www.tux.org/lkml/#export-tainted for information about
tainted modules
Module Pci9030 loaded, with warnings

Getting Module major number..... Ok (MajorID = 254)
Creating device node path....... Ok (Path = /dev/plx)
Creating device nodes........... Ok

Module load complete.



[root@PrimeImplicant root]# exit
exit
[pci@PrimeImplicant bin]$ cd
[pci@PrimeImplicant pci]$ cd tools
[pci@PrimeImplicant tools]$ ln -s /home/pci/PlxLinux/include/ PlxInclude
[pci@PrimeImplicant tools]$ make clean
rm -rf *.o md5test PlxManager manager KernelMan KMTestfile
[pci@PrimeImplicant tools]$ make md5test
[CC] plxinterface.c
[CC] md5test.c
[LD] md5test
[pci@PrimeImplicant tools]$ exit

Script done on Sun 12 Dec 2004 07:37:32 PM MST
```

APPENDIX B (source code):

FIRMWARE:

bit.h

```
#ifndef _IS_INCLUDED_BIT_H
#define _IS_INCLUDED_BIT_H


#define BIT(x) (1<<x)
#define SET_BIT(port, x) port |= BIT(x)
#define CLR_BIT(port, x) port &= (~BIT(x))
#define TOG_BIT(port, x) port ^= BIT(x)


/*control signal port bit locations*/
#define ADS# 0
#define LWE# 1
#define UWE# 2
#define RW# 3
#define OE# 4


#endif
```

bus.h

```
#ifndef _IS_INCLUDED_BUS_H
#define _IS_INCLUDED_BUS_H


#include <iom128v.h>
#include "types.h"


// PE0 -- ADS#
// PE1 -- LW#
// PE2 -- UW#
// PE3 -- RW#
// PE4 -- OE#
// PE5 -- CE#

#define SET_ADS_LOW()         PORTE &= 0xFE
#define SET_ADS_HIGH()        PORTE |= 0x01

#define SET_LW_LOW()          PORTE &= 0xFD
#define SET_LW_HIGH()         PORTE |= 0x02
```

```
#define SET_UW_LOW()          PORTE &= 0xFB
#define SET_UW_HIGH()         PORTE |= 0x04

#define SET_RW_LOW()          PORTE &= 0xF7
#define SET_RW_HIGH()         PORTE |= 0x08

#define SET_OE_LOW()          PORTE &= 0xEF
#define SET_OE_HIGH()         PORTE |= 0x10

#define SET_CE_LOW()          PORTE &= 0xDF
#define SET_CE_HIGH()         PORTE |= 0x20

#define SET_DATAPORT_IN()     DDRC = DDRD = 0x00
#define SET_DATAPORT_OUT()    DDRC = DDRD = 0xFF



uint8 readByte(uint16 addr);
void writeByte(uint16 addr);

uint16 readWord(uint16 addr);
void writeWord(uint16 addr, uint16 word32);

uint32 readLword(uint16 addr);
void writeLword(uint16 addr, uint32 word32);

uint16 translate(uint16 addr);

void buscpy(uint8 *dest, uint8 *src, uint16 size);

#endif
```

bus.c

```
#include "bus.h"
#include <macros.h>


//***************** READ_WORD ******************************
uint8 readByte(uint16 addr)
{
    /*
        PLX chip spans the data across the two memory chips.
        Lower 16 bits go to the first SRAM, upper 16 bits
        go to the second SRAM. We can use the bit 1 of the addr
        to select which memory we are reading from
      */


      uint8 dataIn;
      uint8   byteSel = addr & 0x03;
      addr = addr>>2;

      SET_DATAPORT_IN();
      PORTA = addr;
      PORTB = addr>>8;
      SET_RW_HIGH();
      SET_ADS_LOW();
      SET_OE_LOW();


      if (0 == byteSel) {
          SET_UW_HIGH();
            SET_LW_LOW();
            SET_CE_LOW();
            NOP();NOP();NOP();
            dataIn = PINC;
            SET_CE_HIGH();
      } else if (0x1 == byteSel) {
            SET_UW_HIGH();
            SET_LW_LOW();
            SET_CE_LOW();
            NOP();NOP();NOP();
            dataIn = PIND;
            SET_CE_HIGH();
      } else if (0x2 == byteSel) {
          SET_LW_HIGH();
            SET_UW_LOW();
            SET_CE_LOW();
            NOP();NOP();NOP();
            dataIn = PINC;
            SET_CE_HIGH();
      } else {
            SET_LW_HIGH();
            SET_UW_LOW();
            SET_CE_LOW();
            NOP();NOP();NOP();
            dataIn = PIND;
            SET_CE_HIGH();
      }

      SET_OE_HIGH();
      SET_LW_HIGH();
      SET_UW_HIGH();
      SET_ADS_HIGH();

      return dataIn;
}


/***************** WRITE_WORD *****************************/
void writeWord(uint16 addr, uint16 dataOut)
{

      PORTA = (addr>>2);
      PORTB = (addr>>10);
      SET_ADS_LOW();

      SET_OE_HIGH();

      PORTC = dataOut & 0x00FF;
      PORTD = dataOut>>8;
      SET_DATAPORT_OUT();
      SET_RW_LOW();

      if (addr & 0x0002) { // if bit 1 is set, activate SRAM1
          SET_LW_HIGH();
          SET_UW_LOW();
      }
      else {                     // otherwise, activate SRAM0
            SET_UW_HIGH();
          SET_LW_LOW();
      }

      SET_CE_LOW();
      NOP(); NOP();
      SET_CE_HIGH();       // how many clock cycles will this
instruction take?
      SET_ADS_HIGH();

      SET_LW_HIGH();
      SET_UW_HIGH();
      SET_RW_HIGH();

      SET_DATAPORT_IN();
      PORTC = 0x00;
      PORTD = 0x00;
      return;
}


uint16 readWord(uint16 addr) {

      uint16      dataIn;

      SET_DATAPORT_IN();
      SET_RW_HIGH();
      SET_OE_LOW();

      PORTA = addr>>2;
      PORTB = addr>>10;
      SET_ADS_LOW();

      SET_CE_LOW();
      if (0x02 & addr) {
          SET_LW_HIGH();
            SET_UW_LOW();
            NOP();NOP();NOP();
            dataIn = PIND;
```

```
                dataIn = (dataIn<<8)|PINC;
        } else {
              SET_UW_HIGH();
                SET_LW_LOW();
                NOP();NOP();NOP();
                dataIn = PIND;
                dataIn = (dataIn<<8)|PINC;
        }
        SET_CE_HIGH();

        SET_ADS_HIGH();
        SET_OE_HIGH();
        SET_LW_HIGH();
        SET_UW_HIGH();

        return dataIn;

}

uint32 readLword(uint16 addr) {
    /*since we only have a 16-bit bus, we just split this read up into
       *two word reads and combine the values*/

       uint32 word1 = (uint32)readWord(addr+2);
       return ((word1<<16) | readWord(addr));
}

void writeLword(uint16 addr, uint32 word32) {
       /*this is a little harder than reading, but not much*/
       uint16 word0 = word32 & 0x0000FFFF;
       uint16 word1 = word32>>16;
       writeWord(addr, word0);
       writeWord(addr+2, word1);
}


/*this converts a normal, contiguous memory address to the corresponding
 *address that can be correctly read by the 9030*/
uint16 translate(uint16 addr) {
      uint16 ret;
      /*lower three bits don't change*/
      ret = addr & 0x7;
      /*skip 8 bytes*/
      ret = ret | ((addr & 0x08)<<1);
      /*skip in blocks of 32*/
      ret = ret | ((addr & 0x10)<<2);
      /*high order bits are the same*/
      ret = ret | ((addr & ~(0x1f)) << 2);

      return ret;
}

/*copies data starting at the bus addr stored in src to the local RAM
addr stored in dst*/
```

```
void buscpy(uint8 *dest, uint8 *src, uint16 size) {
       uint16 addr = (uint16)src;
       while (size > 0) {
            *dest = readByte(addr);
            dest++;
            addr++;
            size--;
       }
}
```

main.h

```
/*************************************************************************
**************************************************************************
**                                                                      **
**  File:           main.h                                              **
**  Project:        MyProject                                           **
**  Description:    Learning to program in C++                          **
**                                                                      **
**  Author:         John Doe                                            **
**  Date created:   May 18, 2002                                        **
**  Last modified:  May 21, 2002                                        **
**                                                                      **
**************************************************************************
*************************************************************************/

#ifndef MAIN_H
#define MAIN_H


#endif      // end #ifndef MAIN_H
```

main.c

```
/*************************************************************************
**************************************************************************
**                                                                      **
**  File:           main.c                                              **
**  Project:        Senior Project                                      **
**  Description:    MD5 algorithm                                       **
**                                                                      **
**  Author:         Alex, Dave, Shawn, Tom                              **
**  Date created:   Dec 01, 2004                                        **
**  Last modified:  Dec 08, 2004                                        **
**                                                                      **
**************************************************************************
*************************************************************************/
```

```
// ICC-AVR application builder : 11/19/2004 4:18:14 PM
// Target : M128
// Crystal: 16.000Mhz

#include <iom128v.h>
#include <eeprom.h>
#include <macros.h>
#include <string.h>
#include <math.h>
#include "main.h"
#include "types.h"
#include "md5.h"
#include "bus.h"


#define NULL 0x0

/*control flag constants*/
#define     READY_FLAG  0xCAFE
#define     START_FLAG  0xFEED
#define DONE_FLAG 0xDEAD
#define ERROR_FLAG  0xEEEE
/*command constants*/
#define RESET_CMD   0xAAAA
#define APPEND_CMD      0xBBBB
#define FINISH_CMD      0xDDDD
/*communication addresses (byte addr)*/
#define FLAG_ADDR   0x0000
#define CMD_ADDR  0x0002
#define SIZE_ADDR   0x0004

#define DIGEST0_ADDR 0x0008
#define DIGEST1_ADDR 0x000C
#define DIGEST2_ADDR 0x0010
#define DIGEST3_ADDR 0x0014


/*this address is UNTRANSLATED*/
#define DATA_START 0x0020


/*the master context*/
MD5_CTX digestContext;


/*routines for initializing the Atmel part*/
void port_init(void);
void init_devices(void);

/*routines for handling commands from host PC*/
void initCB(void);
void resetDigest(void);
void appendDigest(uint32 size);
void finishDigest(void);


//*********************  M A I N  ****************************
void main(void) {

      uint16 flag, cmd;

      init_devices();
      resetDigest();

      writeWord(FLAG_ADDR, READY_FLAG);

      /*loop forever, doing what we're told*/
      while (1) {

            /*wait for a command*/
            do {
             flag = readWord(FLAG_ADDR);
            } while (flag != START_FLAG);

            /*we got a start flag, so now figure out which command to
start*/
            cmd = readWord(CMD_ADDR);
            if (RESET_CMD == cmd) {
               resetDigest();
            } else if (APPEND_CMD == cmd) {
               appendDigest(readLword(SIZE_ADDR));
            } else if (FINISH_CMD == cmd) {
               finishDigest();
            } else {
               /*we don't understand the command, so signal an error*/
               writeWord(FLAG_ADDR, ERROR_FLAG);
            }
      }
}




// **************** PORT_INIT ******************************
void port_init(void)
{
      PORTA = 0x00;     // addr[7-0]
      DDRA  = 0xFF;     // out
      PORTB = 0x00;     // addr[11-8]
      DDRB  = 0x0F;     // bits 7-4 HighZ; bits 3-0 out
      PORTC = 0x00;     // data[7-0]
      DDRC  = 0x00;     // in
      PORTD = 0x00;     // data[15-8]
      DDRD  = 0x00;     // in
      PORTE = 0x3F;     // 5=CE#, 4=OE#, 3=RW#, 2=UB#, 1=LB#, 0=ADS#
      DDRE  = 0x3F;     // 7-6 HighZ, 5-0 in
      PORTF = 0x00;     // ports F and G not used, HighZ
      DDRF  = 0x00;
      PORTG = 0x00;
      DDRG  = 0x00;
}


// *************** INIT_DEVICES ****************************
void init_devices(void)
{

      CLI();                        // macro defined in
c:\icc\include\macros.h
      XDIV  = 0x00;           // xtal divider
      XMCRA = 0x00;           // external memory
      port_init();

      MCUCR = 0x00;
      EICRA = 0x00;           // extended ext ints
      EICRB = 0x00;           // extended ext ints
      EIMSK = 0x00;
      TIMSK = 0x00;           // timer interrupt sources
      ETIMSK = 0x00;          // extended timer interrupt sources
      SEI();                        // macro defined in
c:\icc\include\macros.h

}


/*routines for handling commands from host PC*/
void resetDigest(void) {
      MD5Init(&digestContext);
      writeWord(FLAG_ADDR, DONE_FLAG);
}

void appendDigest(uint32 size) {
      uint8 buffer[64];
      uint16 addr = DATA_START;
      /*buffer blocks of 64*/
    while (size >= 64) {
          buscpy(buffer, (uint8*)addr, 64);
            size -= 64;
            addr += 64;
          MD5Update(&digestContext, buffer, 64);
    }
      /*finish off any remaining bytes*/
      if (size > 0) {
          buscpy(buffer, (uint8*)addr, size);
            MD5Update(&digestContext, buffer, size);
            size = 0;
      }

      /*signal completion*/
      writeWord(FLAG_ADDR, DONE_FLAG);
```

```
}

void finishDigest(void) {
       MD5Final (&digestContext);
       /*the digest is simply left in the stage values of the context*/
       writeLword(DIGEST0_ADDR, digestContext.state[0]);
       writeLword(DIGEST1_ADDR, digestContext.state[1]);
       writeLword(DIGEST2_ADDR, digestContext.state[2]);
       writeLword(DIGEST3_ADDR, digestContext.state[3]);
       writeWord(FLAG_ADDR, DONE_FLAG);
}
```

md5.h

```
/* MD5.H - header file for MD5C.C
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

#ifndef _IS_INCLUDED_MD5_H
#define _IS_INCLUDED_MD5_H

#include "global.h"



/* MD5 context. */
typedef struct {
  UINT4 state[4];                                   /* state (ABCD) */
  UINT4 count[2];        /* number of bits, modulo 2^64 (lsb first) */
  unsigned char buffer[64];                         /* input buffer */
} MD5_CTX;

void MD5Init PROTO_LIST ((MD5_CTX *));
void MD5Update PROTO_LIST
  ((MD5_CTX *, unsigned char *, unsigned int));
void MD5Final PROTO_LIST ((MD5_CTX *));




#endif
```

md5c.c

```
/* MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data

Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

#include "global.h"
#include "md5.h"

/* Constants for MD5Transform routine.
 */


#define S11 7
#define S12 12
```

```
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST
  ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST
  ((UINT4 *, unsigned char *, unsigned int));
static void MD5_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD5_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = {
  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H and I are basic MD5 functions.
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits.
 */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
Rotation is separate from addition to prevent recomputation.
 */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define GG(a, b, c, d, x, s, ac) { \
 (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define HH(a, b, c, d, x, s, ac) { \
 (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define II(a, b, c, d, x, s, ac) { \
 (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }

/* MD5 initialization. Begins an MD5 operation, writing a new context.
 */
void MD5Init (context)
MD5_CTX *context;                                        /* context */
{
  context->count[0] = context->count[1] = 0;
  /* Load magic initialization constants.
*/
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}

/* MD5 block update operation. Continues an MD5 message-digest
  operation, processing another message block, and updating the
  context.
 */
void MD5Update (context, input, inputLen)
MD5_CTX *context;                                        /* context */
unsigned char *input;                                /* input block */
unsigned int inputLen;                     /* length of input block */
{
  unsigned int i, index, partLen;

  /* Compute number of bytes mod 64 */
  /*this is the number of bytes hanging around in the buffer from the
last go around*/
  index = (unsigned int)((context->count[0] >> 3) & 0x3F);

  /* Update number of bits (64-bit addition) */
  if ((context->count[0] += ((UINT4)inputLen << 3)) < ((UINT4)inputLen
<< 3)) {
       context->count[1]++;
  }
  context->count[1] += ((UINT4)inputLen >> 29);

  /*partLen is the number of bytes needed to finish the 64 byte block
started
   *during the previous run*/
  partLen = 64 - index;

  /* Transform as many times as possible.
*/
```

```
  if (inputLen >= partLen) {
       /*finish off the previously started block*/
     MD5_memcpy((POINTER)&context->buffer[index], (POINTER)input,
partLen);
     MD5Transform (context->state, context->buffer);
       /*do as many full blocks as possible*/
       for (i = partLen; i + 63 < inputLen; i += 64) {
         MD5Transform (context->state, &input[i]);
       }
       /*The remaining incomplete block will get stuck in the buffer
starting at index*/
       index = 0;
  } else {
       /*the input still doesn't provide enough bytes to finish off the
block in the buffer*/
       i = 0;
  }
  /* Buffer remaining input for either the next go around or the
finish*/
  MD5_memcpy((POINTER)&context->buffer[index], (POINTER)&input[i],
inputLen-i);
}

/* MD5 finalization. Ends an MD5 message-digest operation, writing the
  the message digest and zeroizing the context.
 */
void MD5Final (context)
MD5_CTX *context;                                       /* context */
{
  unsigned char bits[8];
  unsigned int index, padLen;

  /* Save number of bits */
  Encode (bits, context->count, 8);

  /* Pad out to 56 mod 64.*/
  /* We only need 56 (or 120) bytes in the last block because we're
going to append 8 bytes*/
  /* that contain the total number of bits digested*/

  index = (unsigned int)((context->count[0] >> 3) & 0x3f);
  padLen = (index < 56) ? (56 - index) : (120 - index);
  MD5Update (context, PADDING, padLen);

  /* Append length (before padding) */
  MD5Update (context, bits, 8);

}

/* MD5 basic transformation. Transforms state based on block.
 */
static void MD5Transform (state, block)
UINT4 state[4];
unsigned char block[64];
```

```
{
  UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

  Decode (x, block, 64);

  /* Round 1 */
  FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
  FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
  FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
  FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
  FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
  FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
  FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
  FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
  FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
  FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
  FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
  FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
  FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
  FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
  FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
  FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

 /* Round 2 */
  GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
  GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
  GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
  GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
  GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
  GG (d, a, b, c, x[10], S22, 0x02441453); /* 22 */
  GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
  GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
  GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
  GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
  GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
  GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
  GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
  GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
  GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
  GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

  /* Round 3 */
  HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
  HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
  HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
  HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
  HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
  HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
  HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
  HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
  HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
  HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
  HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
  HH (b, c, d, a, x[ 6], S34, 0x04881d05); /* 44 */
  HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
  HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
  HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
  HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

  /* Round 4 */
  II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
  II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
  II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
  II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
  II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
  II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
  II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
  II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
  II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
  II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
  II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
  II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
  II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
  II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
  II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
  II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

  state[0] += a;
  state[1] += b;
  state[2] += c;
  state[3] += d;

  /* Zeroize sensitive information.
   */
  MD5_memset ((POINTER)x, 0, sizeof (x));
}

/* Encodes input (UINT4) into output (unsigned char). Assumes len is
  a multiple of 4.
 */
static void Encode (output, input, len)
unsigned char *output;
UINT4 *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4) {
 output[j] = (unsigned char)(input[i] & 0xff);
 output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
 output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
 output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
  }
}

/* Decodes input (unsigned char) into output (UINT4). Assumes len is
  a multiple of 4.
 */
```

```
static void Decode (output, input, len)
UINT4 *output;
unsigned char *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4)
 output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
   (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Note: Replace "for loop" with standard memcpy if possible.
 */

static void MD5_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
    output[i] = input[i];
}

/* Note: Replace "for loop" with standard memset if possible.
 */
static void MD5_memset (output, value, len)
POINTER output;
int value;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 ((char *)output)[i] = (char)value;
}
```

types.h

```
/***********************************************************************
************************************************************************
**                                                                    **
**  File:           types.h                                           **
**  Project:        MyProject                                         **
**  Description:    Learning to program in C++                        **
**                                                                    **
**  Author:         John Doe                                          **
**  Date created:   May 18, 2002                                      **
**  Last modified:  May 21, 2002                                      **
**                                                                    **
************************************************************************
***********************************************************************/

#ifndef     TYPES_H
#define TYPES_H

typedef unsigned char       uint8;
typedef unsigned short      uint16;
typedef unsigned long       uint32;

typedef signed   char       int8;
typedef signed   short      int16;
typedef signed   long       int32;


#endif
```

SOFTWARE:

Makefile

```
DEFINES = -DPLX_LINUX -DPCI_CODE -DPLX_LITTLE_ENDIAN
CFLAGS = -c
INCLUDES = -I. -I./include -I./PlxInclude
LIBS = $(LIBDIR)/PlxApi.a $(LIBDIR)/PciDrvApi.a -lcrypto
LIBDIR = lib
SRCDIR = src

OBJS = plxinterface.o md5test.o
APPS = md5test PlxManager manager KernelMan KMTestfile

#targets
all: $(APPS)

md5test: $(OBJS)
      @echo [LD] md5test
      @gcc $(OBJS) $(LIBS) -o md5test

PlxManager: PlxManager.o
      @echo [LD] PlxManager
      @gcc PlxManager.o $(LIBS) -o PlxManager

manager: manager.o plxinterface.o
      @echo [LD] manager
      @gcc manager.o plxinterface.o $(LIBS) -o manager

KernelMan: KernelMan.o plxinterface.o
      @echo [LD] KernalMan
      @gcc KernelMan.o plxinterface.o $(LIBS) -o KernelMan

KMTestfile: KMTestfile.o plxinterface.o
```

```
	@echo [LD] KMTestfile
	@gcc KMTestfile.o plxinterface.o $(LIBS) -o KMTestfile

clean:
	rm -rf *.o $(APPS)

%.o: $(SRCDIR)/%.c
	@echo [CC] $*.c
	@gcc $(DEFINES) $(CFLAGS) $(INCLUDES) -o $@ $<
```

manager.c

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <string.h>
#include <openssl/md5.h>
#include "plxinterface.h"
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>



__volatile__ unsigned char *barAddr;

//************************ M A I N ****************************
int main(int argc, void *argv[]) {
	int temp, len; /*file;*/
	int blockCount, totalBlockCount;
	MD5_CTX	digestContext;
	int hostDigest[4];
	int cardDigest[4];
	//	char buffer[MAX_CARD_DATA_SIZE];
	char buf[8161];
	int sock, msgsock;
	int length;
	int rval;
	struct sockaddr_in server;
	struct sockaddr_in client;
	struct hostent *cname;
	int port;
	int timesAccessed=0;


	barAddr = (unsigned char *)openPlxDevice();
	if (0 == barAddr) {
		exit(-1);
	}
```

```
////////////////////
//Wait for card to get ready

printf("Waiting for ready signal\n");
while ((readFlag() != READY_FLAG) && (readFlag() != DONE_FLAG));

/*  Create socket  */
if ( (sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {   perror("opening steam socket");
  exit(1);
  }

/*  Name socket using wildcards  */
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
port = 2080;
server.sin_port = port;
if (bind(sock, (struct sockaddr *)&server, sizeof(server)) < 0)
  {   perror("binding stream socket");
  exit(1);
  }
listen(sock, 10);
while (1) {


  /*  Start accepting connections  */

  length = sizeof(client);
  msgsock = accept(sock, (struct sockaddr *) &client, &length);
  cname = gethostbyaddr((char *) &client.sin_addr, 4, AF_INET);
  if (msgsock < 0)
    {
      perror("accept");
    }
  int size;
  rval = recv(msgsock,(char *)&size, 4, 0);
  if (rval!=4) {
    perror("bad size");
    exit(-1);
  }
  timesAccessed++;
  totalBlockCount = (size/MAX_CARD_DATA_SIZE)+1;

  ////////////////////
  //Reset Operation

  MD5_Init(&digestContext);

  printf("Resetting digest\n");
  sendCommand(RESET_CMD, 0);
  printf("size:%d\n",size);
  while (readFlag() != DONE_FLAG);
```

```
        ///////////////////
        //Perform Digestion

        blockCount = 0;
        while (size>0)
          {
            //          printf("do we get in the while loop?\n");
            if (size<MAX_CARD_DATA_SIZE)
            {

              rval = recv(msgsock,buf,size,0);

              len = size;
              size=size-size;
            }
            else
            {
              //  printf("Before Recv\n");
              rval = recv(msgsock,buf,MAX_CARD_DATA_SIZE,0);
              //printf("after Recv\n");
              len = MAX_CARD_DATA_SIZE;
              size = size-MAX_CARD_DATA_SIZE;
            }
            //printf("len is: %d \n",len);
            //printf("Here's what ended up in the string:\n%s\n",buf);
            MD5_Update(&digestContext, buf, len);

            blockCount++;
            printf("Processing block %d/%d (size = %d)\n", blockCount,
totalBlockCount, len);
            copyToCard(barAddr+DATA_OFFSET, buf, len);
            sendCommand(APPEND_CMD, len);

            /*wait for card to finish*/
            while (readFlag() != DONE_FLAG);
          } /*end of while loop*/

            //////////////
            //Finish Digestion

        MD5_Final((char*)hostDigest, &digestContext);

        printf("Finishing digest\n");
        sendCommand(FINISH_CMD, 0);
        while (readFlag() != DONE_FLAG);

        /*read in finished digest*/
        copyFromCard((char*)cardDigest, barAddr+DIGEST_OFFSET, 16);

        //////////////
        //Compare results

        printf("\n");
        printf("Host Digest: %08x %08x %08x %08x\n", hostDigest[0],
hostDigest[1],
             hostDigest[2], hostDigest[3]);
        printf("Card Digest: %08x %08x %08x %08x\n", cardDigest[0],
cardDigest[1],
             cardDigest[2], cardDigest[3]);
        if ( !memcmp(hostDigest, cardDigest, 16) ) {
          printf("SUCCESS!\n");
        } else {
          printf("FAILURE!\n");
        }
        send(msgsock,&cardDigest,16,0);
        close(msgsock);

      }/*end of while for recieving data*/

      close(sock);
      return 0;

}  /* main() */
```

md5test.c

```
/*
**    Example of receiving data using a stream socket
**  This program creates a stream socket, indicates the portnumber
**  assigned, waits for a connection, and prints out messages from
**  that connection.
*/

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <string.h>
#include <openssl/md5.h>
#include "plxinterface.h"


__volatile__ unsigned char *barAddr;

//************************* M A I N *****************************
int main(int argc, void *argv[]) {
      int temp, len, file;
      MD5_CTX     digestContext;
      int hostDigest[4];
      int cardDigest[4];
      char buffer[MAX_CARD_DATA_SIZE];

      /*try to open a file for reading*/
      if (argc == 2) {
            file = open((char*)argv[1], O_RDONLY);
            if (-1 == file) {
                  printf("Failed to open file for reading\n");
                  exit(-1);
            }
      } else {
        printf("You must specify a the name of a file to digest\n");
            exit(-1);
      }

      /*now try to open the plx device*/
      barAddr = (unsigned char *)openPlxDevice();
      if (0 == barAddr) {
            exit(-1);
      }


////////////////////
//Wait for card to get ready

    printf("Waiting for ready signal\n");
    while ((readFlag() != READY_FLAG) && (readFlag() != DONE_FLAG));


////////////////////
//Reset Operation

      MD5_Init(&digestContext);

      printf("Resetting digest\n");
      sendCommand(RESET_CMD, 0);
    while (readFlag() != DONE_FLAG);

///////////////////
//Perform Digestion

      /*send entire file*/
      len = read(file, buffer, MAX_CARD_DATA_SIZE);
      while (len > 0) {
            MD5_Update(&digestContext, buffer, len);

            printf("Processing block (size = %d)\n", len);
            copyToCard(barAddr+DATA_OFFSET, buffer, len);
            sendCommand(APPEND_CMD, len);

            /*wait for card to finish*/
            while (readFlag() != DONE_FLAG);

            /*read next block*/
            len = read(file, buffer, MAX_CARD_DATA_SIZE);
      }

//////////////
```

```
//Finish Digestion

      MD5_Final((char*)hostDigest, &digestContext);

      printf("Finishing digest\n");
      sendCommand(FINISH_CMD, 0);
      while (readFlag() != DONE_FLAG);

      /*read in finished digest*/
      copyFromCard((char*)cardDigest, barAddr+DIGEST_OFFSET, 16);

//////////////
//Compare results

      printf("\n");
      printf("Host Digest: %08x %08x %08x %08x\n", hostDigest[0],
hostDigest[1],

hostDigest[2], hostDigest[3]);
      printf("Card Digest: %08x %08x %08x %08x\n", cardDigest[0],
cardDigest[1],

cardDigest[2], cardDigest[3]);
      if ( !memcmp(hostDigest, cardDigest, 16) ) {
            printf("SUCCESS!\n");
      } else {
            printf("FAILURE!\n");
      }



    return 0;

}  /* main() */
```

plxinterface.h

```
#ifndef _IS_INCLUDE_PLXINTERFACE_H
#define _IS_INCLUDE_PLXINTERFACE_H



#define FLAG_OFFSET 0x00
#define CMD_OFFSET 0x02
#define SIZE_OFFSET 0x04
#define DIGEST_OFFSET 0x08
#define DATA_OFFSET 0x20

#define READY_FLAG 0xCAFE
#define START_FLAG 0xFEED
#define DONE_FLAG 0xDEAD
```

```
#define ERROR_FLAG 0xEEEE

#define RESET_CMD 0xAAAA
#define APPEND_CMD 0xBBBB
#define FINISH_CMD 0xDDDD


#define MAX_CARD_DATA_SIZE 8160




void copyToCard(__volatile__ char *dst, char *src, int len);
void copyFromCard(char *dst, __volatile__ char *src, int len);
void sendCommand(unsigned short command, unsigned int size);
unsigned short readFlag(void);
int openPlxDevice(void);


#endif  /*redef guard*/
```

plxinterface.c

```
#include <PlxApi.h>
#include "plxinterface.h"


/*this is a global defined somewhere else*/
extern __volatile__ unsigned char *barAddr;

/*attempts to open a PLX 9030 device and map PCI address space
 *returns the address of the PCI BAR on success, 0 on failure
*/

int openPlxDevice(void) {
    U32  deviceNum = 0;
      int ret = 0;
    HANDLE hDevice;
    DEVICE_LOCATION deviceLoc;

    deviceLoc.BusNumber       = (U8)-1;
    deviceLoc.SlotNumber      = (U8)-1;
    deviceLoc.VendorId        = PLX_VENDOR_ID;
    deviceLoc.DeviceId        = PLX_9030RDK_LITE_DEVICE_ID;
    deviceLoc.SerialNumber[0] = '\0';

    if (PlxPciDeviceFind(&deviceLoc, &deviceNum) != ApiSuccess) {
            printf("\nERROR: Unable to find any PLX 9030 device\n");
        return 0;
    }
    if (ApiSuccess != PlxPciDeviceOpen(&deviceLoc, &hDevice))    {
        printf("\nERROR: Unable to open PLX device\n");
        return 0;
    }

    if (ApiSuccess != PlxPciBarMap(hDevice, 2, &ret)) {
      printf("\nERROR: Unable to map PCI BAR 2\n");
      return 0;
    }
      return ret;
}

void copyToCard(__volatile__ char *dst, char *src, int len) {
      char b = *dst;
      while (len>0) {
            *dst = *src;
            dst++; src++;
            len--;
            /*every 16 bytes perform a read*/
            /*this forces the PLX chip to flush its write buffers*/
            if (0 == len & 0x0f) {
                  b = *dst;
            }
      }
}

void copyFromCard(char *dst, __volatile__ char *src, int len) {
      while (len>0) {
            *dst = *src;
            dst++; src++; len--;
      }
}

void sendCommand(unsigned short command, unsigned int size) {
      char c;
      *((unsigned short*)(barAddr+CMD_OFFSET)) = command;
      *((unsigned int*)(barAddr+SIZE_OFFSET)) = size;
      *((unsigned short*)(barAddr+FLAG_OFFSET)) = START_FLAG;
      /*perform a read to force a buffer flush*/
      c = (*barAddr+7)>>command;
}

unsigned short readFlag(void) {
      return *((unsigned short*)(barAddr+FLAG_OFFSET));
}
```

Testfile.c

```
/*
**    Example of a client program sending information to a server.
**  This program creates a socket.  It then asks for a machine and port
**  to connect to.  Once the connection is made, a message is sent, a
**  reply is read, and it loops.
*/

#include <stdio.h>
#include <string.h>
#include <sys/types.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <fcntl.h>


main(argc, argv)
int argc;
char *argv[];

{      int sock;
       struct sockaddr_in server;
       struct hostent *hp;
       char buf[8192];
       int len;
       int rval;
       int fileLength;
       int fd;
       /*try to open a file for reading*/
       if (argc == 2) {
             fd = open((char*)argv[1], O_RDONLY);
             if (-1 == fd) {
                   printf("Failed to open file for reading\n");
                   exit(-1);
             }
       } else {
         printf("You must specify a the name of a file to digest\n");
             exit(-1);
       }
       fileLength = lseek(fd,0,SEEK_END);
       lseek(fd,0,SEEK_SET);


/*  Create socket  */
       sock = socket(AF_INET, SOCK_STREAM, 0);
       if (sock < 0)
       {     perror("opening stream socket");
             exit(1);
       }

/*  Connect socket  */
       server.sin_family = AF_INET;
       //buf = "localhost";
       hp = gethostbyname("localhost");
       if (hp == 0)
```

```
      {     fprintf(stderr, "%s: unknown host\n", buf);
            exit(1);
      }
      memcpy((char *)&server.sin_addr, (char *)hp->h_addr, hp-
>h_length);
      //buf = "8200";
      server.sin_port = 2080;//htons((short) atoi(buf));


/* Send data until we get a blank line  */

            if (connect(sock, (struct sockaddr *) &server,
sizeof(server)) < 0)
            {     perror("connecting stream socket");
                  exit(1);
            }
      send(sock,(char *)&fileLength,4,0);
      printf("sent the file length.\n");
      len = read(fd, buf, 8160);
      printf("read the first part of the file\n");
      while (len > 0) {
            rval = send(sock, buf, len, 0);
            printf("sent %d bytes\n",rval);
            len = read(fd,buf,8160);
      }
      //    printf("did we finish writing?\n");
      recv(sock,buf,16,0);
      //buf[16] = '\0';
      printf("Digest received: '%08x %08x %08x %08x'\n",*(int
*)(buf),*(int *)(buf+4), *(int *)(buf+8), *(int *)(buf+12));
      close(sock);

      return 0;
}  /* main() */
```

Teststrings.c

```
/*
**    Example of a client program sending information to a server.
**  This program creates a socket.  It then asks for a machine and port
**  to connect to.  Once the connection is made, a message is sent, a
**  reply is read, and it loops.
*/




#include <stdio.h>
#include <string.h>
#include <sys/types.h>
```

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>



main(argc, argv)
int argc;
char *argv[];

{     int sock;
      struct sockaddr_in server;
      struct hostent *hp;
      char buf[1024];
      int len;
      int rval;


      if (argc != 1)
        {   fprintf(stderr,"%s has no command-line arguments\n",
argv[0]);
        exit(1);
        }



      /* Send data until we get a blank line  */
      while (1) /* until null input */
        {
          /*  Create socket  */
          sock = socket(AF_INET, SOCK_STREAM, 0);
          if (sock < 0)
            {
            perror("opening stream socket");
            exit(1);
            }

          /*  Connect socket  */
          server.sin_family = AF_INET;
          //buf = "localhost";
          hp = gethostbyname("localhost");
          if (hp == 0)
            {     fprintf(stderr, "%s: unknown host\n", buf);
            exit(1);
            }
          memcpy((char *)&server.sin_addr, (char *)hp->h_addr, hp-
>h_length);
          //buf = "8200";
          server.sin_port = 2080;//htons((short) atoi(buf));
          printf("Enter string to send, just return to end: ");
          gets(buf);
          len = strlen(buf);
          if (len == 0)  break;
          if (connect(sock, (struct sockaddr *) &server, sizeof(server))
< 0)
            {
            perror("connecting stream socket");
            exit(1);
            }
          send(sock,(char *)&len,4,0);
          if (send(sock, buf, len, 0) < 0)
            {
            perror("writing on stream socket");
            exit(1);
            }
          rval = recv(sock, buf, 16, 0);
          buf[17] = '\0';
          printf("Digest received '%08x %08x %08x %08x'\n",*((int
*)buf),*((int *)(buf+4)),*((int *)(buf+8)),*((int *)(buf+12)));
          close(sock);

        }
      close(sock);
      return 0;
}  /* main() */
```