Computerized Instrumentation Design

PHY 215

Prepared by Kerry K. Kuehn,
Wisconsin Lutheran College

March 25, 2019

Contents

List of Figures

List of Tables

Overview

Introduction

This manual is designed to accompany an introductory course on the uses of a small computer in the laboratory. The aim of the course is to teach you how to set up a computer—controlled laboratory experiment "from scratch" and produce a technical paper. Specifically, we will cover the following topics: installation and system administration of a UNIX-like operating system, elementary computer programming in "C", input and output ports, analog-to-digital and digital-to-analog conversion, digital signal processing, thermistors, temperature control, least squares fitting to experimental data, heat capacity or thermal conductivity measurements, data analysis, and scientific documentation.

During a traditional 15-week term, the course meets once per week for a one hour lecture and twice per week for three hour laboratory sessions. During a compact, 12-day term (e.g. January-term), the course meets daily for seven hour laboratory sessions. Four hours will be supervised by the instructor, three hours are unsupervised.

Course grades will be based on (i) the exercises completed in your laboratory notebook and (ii) a final paper. Your laboratory notebook should be handed in to the instructor for grading immediately upon completing each exercise. Your final paper is due at the end of the semester, and should clearly, concisely, and completely, describe the final project of the course.

Students enrolled in the course typically have a wide range of backgrounds. Some students come with only a strong desire to learn how computers work; other students, because of their previous knowledge, could practically teach the course. For this reason, the course is designed to be rather flexible in that students are allowed to work at their own pace. Since the course typically has less than seven students, I encourage those of you who are more experienced to share your knowledge and wisdom with students who have not had as much preparation. Be generous! I think you will find that in explaining concepts to others, you will deepen your own understanding as well.

Motivation

Why use computerized instruments in the laboratory? Primarily, for automation. Automation, first, allows for reduction of errors in data collection. It is true that an experimenter can read an instrument which tells him the temperature of a piece of material and record it in his laboratory notebook, but when he must do so ten or a hundred or a thousand times, the chance of a scribal error increases to often an unacceptable level. Automation also reduces the amount of human labor. The experimenter can do other tasks, often remotely, while his experiment is running, rather than focusing all of his time on repetitive tasks in the laboratory. In addition, automation allows for performance which is unattainable otherwise. For instance, a human being could not record a hundred temperature readings per second; a computer can. Finally, automation allows for information to be readily stored in digital form, which is far easier to handle than in paper-written form.

A word of warning about automation, however, is in order. Automation can often introduce systematic errors which, if not detected and corrected, can prove catastrophic. Consider the crash of NASA's Mars Climate Orbiter in September of 1999: a programming error went undetected and a $330 million dollar science project crashed in an entirely automated fashion onto the surface of Mars. The adage that a computer is only as smart as its designer is entirely appropriate. Automation is not a silver bullet and should not be thought of as such. A great deal of planning and careful experimentation is necessary to produce a reliably automated system. Hopefully, this course will give you a deeper understanding of computer technology so that you may intelligently use it to your advantage.

Equipment

We will be using PCs, or personal computers, as opposed to "workstations", for laboratory automation. These are also often called "Wintel" machines since their motherboard was developed around the Intel processor architecture and they often use a Microsoft Windows-like operating system. These PCs are descendants of the original IBM-PC.

You will notice that in this course, we are using rather old computers. In fact, you may feel a bit like you are tinkering around on an old automobile. This is intentional—I want you to feel very comfortable with exploring, without having to worry about breaking a costly new piece of equipment. These computers are inexpensive, so have fun!

The PC which you will be using consists of

- a motherboard containing the microprocessor and a number of other integrated circuit chips (comprising the "chip-set"),
- a power supply,
- a CD-ROM drive,

- a hard disk drive,
- perhaps a floppy disk drive,
- an ethernet card (probably inserted into a PCI slot in your motherboard,
- at least one unused ISA slot,
- a monitor,
- a keyboard, and
- a mouse.

Into the ISA expansion slot on your PC's motherboard you will be installing an interface board, or A/D (A-to-D) board. The interface board is similar to your keyboard, mouse, and monitor, in that it allows your microprocessor to communicate with the outside world. It is unlike these three peripheral devices in that it is not designed to communicate with a human, but rather with a piece of equipment that produces a voltage (*e.g.* the signal from a thermistor) or needs an electronic signal to operate (*e.g.* a light emitting diode).

In addition to the computer and interface board, you will be using auxiliary electronic equipment. On a proto-board (prototyping board), you will be constructing simple circuits using resistors, capacitors, light emitting diodes (LEDs), field effect transistors (FETs), thermistors, and switches. To operate and test these circuits, you will also need an oscilloscope, a function generator, and a DC power supply.

Finally, you will need a small block of aluminum to test your temperature measurement and control techniques and a rod of copper to measure its heat capacity and thermal conductivity as functions of temperature (more on this later...).

Laboratory Notebook

Each of the exercises performed in this course should be clearly documented in your laboratory notebook. If you have a neatly organized laboratory notebook, you can quickly and easily refer to your previous work for guidance when you later come upon a similar problem. To this end, your notebook should be large enough that you can tape printouts of programs which you have written and graphs of data which you have generated onto the pages of the notebook. You should begin each entry in the laboratory notebook with the current date. Furthermore, I suggest that you not try to conserve paper in your notebook by cramming things together; leave plenty of room between paragraphs so that you (and the instructor) can later make comments on your work; a well organized laboratory notebook is far more valuable than the paper on which it is written. And you can always get another notebook.

Course Literature

There is no single text for this course; we will primarily follow the exercises outlined in this manual. To a large extent, this manual is based on undergraduate courses taught at at the University of California at Santa Barbara and at Cornell University. Some sections of this manual are adapted directly from *IBM PC in the Laboratory*, by Thompson and Kuckes[6], the text formerly used at Cornell. You will need to purchase a copy of *The C Programming Language*, 2nd Edition, by Kernighan and Ritchie[2], which is a (perhaps too) succinct book on programming in C. Information about the QNX Realtime Platform may be found by using the help menu on the QNX graphical user interface, or on the QNX website at http://www.qnx.com. A good overview of real-time operating systems and embedded programming can be found in *An Embedded Software Primer* by David Simon[5]. *Upgrading and Repairing PCs*, by Scott Mueller[4], and *The Art of Electronics*, by Horowitz and Hill [1], are classic texts covering personal computers and electronics respectively. For the latest information about computer hardware and software, you may look at various trade magazines. Of course a web search for topics such as "QNX", "computer architecture" or "data acquisition" will also provide you with a great deal of information.

Chapter 1

System Administration

1.1 The Operating System

Without an operating system (OS), your computer is merely a collection of metal and plastic and semiconductor parts. The operating system allows a user to manage a computer's resources—its disk drive, monitor, keyboard, application programs, network and sound cards, and so on. Examples of popular operating systems are Linux, UNIX, Mac OSX, and Microsoft Windows.

In this course, we will be using an operating system called QNX (specifically, QNX Realtime Operating System, version 6.1). It is a Unix-like real-time operating system developed by Gordon Bell and Dan Dodge when they were students at the University of Waterloo in 1980. QNX is similar to UNIX in the sense that the user has access to a command line interface, as opposed to strictly a graphical user interface (GUI), which has been popularized by Macintosh and Microsoft. Furthermore, the commands used in QNX are very much like those used in UNIX (more about this later). If you are used to Microsoft operating systems and this business all seems a bit foreign to you, you should not be concerned. It will be a cinch after a bit of experience.

QNX is a popular operating system for computers which control time—critical processes. For instance, it has been (or is being) used by

- the New York Stock Exchange for security,
- Texaco for offshore oil-rig monitoring,
- NASA for controlling the mechanical arm in the cargo bay of the Space Shuttle,
- in the automotive industry, for navigation, infotainment and telematics,
- smartphones and mobile devices,
- medical technology,

- and even casino gaming systems.

QNX is a flexible microkernel-based operating system (as opposed to other operating systems with a monolithic kernel). Since most of QNX runs as small and independent processes which communicate *via* message passing, the various processes can be turned on or off at will. Thus, it can be scaled down so as to fit in the limited memory of a calculator, or scaled up to control a complex network of computers running an assembly line in a factory. We will be using QNX to monitor and control experiments in the physics laboratory. It is our hope that in becoming familiar with QNX, you will gain a deeper understanding of operating systems in general.

1.1.1 PC Setup and QNX Installation

1. Let us first assemble the computer system which we will be using. You should have a monitor, a mouse, a keyboard, and a computer chassis containing a motherboard, a graphics card, a hard drive, a floppy drive, a CD-ROM player, a power supply, and a cooling fan. Make sure the mouse, keyboard, and monitor are plugged into the correct ports of the PC and that the power cord is plugged in.

2. Now let us install the operating system. We first need to make the computer boot from the CD-ROM drive, rather than from the hard drive. First, reboot your computer by turning it off, waiting ten or fifteen seconds, then turning it back on again. Press the F1 key when prompted to enter the BIOS Setup Utility.

3. Select the CD-ROM Drive as the first boot device from the Boot menu. (Many older computers are not capable of booting directly from the CD-ROM drive, but your BIOS should enable you to do so.) You may need to use the ENTER, ESC, and the +, -, and arrow keys to change settings in the BIOS.

4. While you are at it, spend a few minutes snooping around in the BIOS Setup Utility (especially if you have never done so). Check your system date and time settings and make sure they are correct. What are some of the other interesting things that you can do? When you are finished, exit, and make sure you save the changes you made.

5. Insert the QNX Realtime Platform CD in the CD-ROM drive and reboot the computer. When prompted, press the space-bar to input boot options. The specific sequence of commands you must enter may depend on the present configuration of your system; the following is a guide.

6. Select F6 so that the computer is verbose about what it is doing during installation. Select F3 to install QNX from your CD. Again select F2, the verbose setting, if allowed. You will probably need to accept a QNX Software agreement. Select F1 to install to "/dev/hd0", which is the

computer's hard drive; F1 to install from "cd0", which is the CD-ROM drive; F2 to keep the size less than 8.4 GB of the hard disk to a QNX partition. F2 to wipe out what ever else you had on the disk. Install the QNX partition boot loader. You will now need to wait while a number of files are copied from the CD to the new QNX partition on your hard disk. After it is done, your system will automatically reboot. Be sure to now remove the CD from the CD-ROM drive before it reboots.

7. After your system (automatically) reboots, a screen will appear asking you to verify the video card which is detected at boot. Be sure to deselect the "Always boot into photon graphical environment" box.[1] Then select the "Test" option to test your video driver settings. Finally, hit "continue." When prompted to login, hit "shutdown" and exit to text mode. You can now log in using the command-line interface as *root* without using a password.

1.1.2 System Capabilities

1. How much memory does your hard disk have? You may need to look into the BIOS again if you do not remember. What is your processor speed? How much RAM (Random Access Memory) does your computer have?

2. About how many copies of the book of Genesis would fit on your hard disk? How many copies would fit on a floppy disk? On a CD? On a USB stick? (Hint: To answer these questions, you will need to estimate the memory required per ASCII character, and how many characters are on each page...)

3. How might you run Windows and QNX on the same computer at different times?

1.2 Using the Command Line & GUI

Congratulations! You have successfully converted a bunch of plastic, metal, and semiconductor parts into a useful system. Now let us learn a bit about QNX. Each user of a computer typically has a personal space–a "login directory"–in which his files are located. This directory is protected by a password so that no one else may access his files. There is, however, a special directory called "root", the user of which may access all files on the computer. Typically only one person, the "super-user" (usually the system administrator), knows the root password. There is great power, and hence great responsibility, associated with being the "super-user." You may change, and correspondingly mess up,

[1] This creates a file called "nophoton" in the directory "etc/config/system." You may later wish to delete this file, but for the time being, we want to get a better feel for how QNX works without a graphical user interface (GUI). Also, QNX runs noticeably faster without the GUI running.

everything in the system. To avoid the burden of this great responsibility, you should now create a personal directory for yourself.

1.2.1 Setting up a user account

1. After having installed QNX, you are logged in as the "superuser". To set the root password, type

   passwd root

   Let us all choose "phy_215" as the root password. Now to create a personal directory, type

   passwd loginname

   where loginname is a name of your choice, usually your first name. After hitting the ENTER key, you will be asked a number of questions. You should assign a user ID to yourself, larger than 100, which is not used by anyone else. Type

   cat /etc/passwd

   to see which, if any, user IDs have already been used. You will also need to select a group. For convenience, simply select Group 100. You will also need to select a password. Pick one which includes one or two characters such as _, +, -, ., &, etc. so as to make it more difficult for people running password-cracking programs to infiltrate your system.

2. Now we'd like to see how the system looks when the power comes on. Type the command

   shutdown

   After the computer shuts down, wait ten or fifteen seconds, then turn it back on and watch it as it reboots. This is called a "cold boot" since we turned the power off. After a minute or so, the computer will ask you to log in saying:

   login:

   Answer with a login name followed by the ENTER key. At the next prompt, type in your password and the ENTER key. You should now see the QNX command line prompt

   $

   Now type

   pwd

   This is a QNX command which will tell you your present working directory:

   /home/loginname

   If this works, you are logged into your personal directory. You cannot harm anything else on the system, so relax. We will have the opportunity to learn many more commands.

You have now been introduced to a few QNX commands: **cat**, **shutdown**, **pwd** and **passwd**. If you are done, you should do an orderly shutdown of the system. Don't forget to turn off the monitor as well! If you plan to work some more later on the same day, leave the power on, but do log out so no one can come along and mess up your files.

Now here is a list of some more commonly used commands provided by QNX

pwd	print working directory, gives current location
cd	changes directory to your home directory
cd ..	moves up one level in the directory structure to the parent directory
cd *path*	moves you to the directory specified in path
ls -F	lists contents of current directory, distinguishing files from subdirectories
cat *file*	types contents of named file to the monitor
use *command*	on-line help giving syntax and command line options
mkdir *dir*	creates a subdirectory
rmdir *dir*	deletes a subdirectory
echo *text*	echoes text to the monitor

Let's look at how to use some of these commands.

1.2.2 Navigation through Directories

1. Be sure you are logged in under your user account (not as root). Go to the top level directory by typing

   cd /

   Determine the contents of the top level directory by typing

   ls -F

   Notice that **ls** -f will not work since QNX is case sensitive. Try some other options such as **ls** -a and **ls** -l. Type **use ls** to see what options are available with the **ls** command.

2. Describe at least one of the **ls** options not already mentioned above and which you think might be useful in your notebook.

3. What files are contained in /etc/rc.d? Distinguish clearly between files and subdirectories. To explore the contents of these files, you can use

   cat *filename*

   This will print the contents of the file on the monitor. If you cannot read the whole file, you may type

   more *filename*

   Hitting the spacebar puts more text on the monitor. A carriage return advances one line. Typing b moves you back one screen. In order to quit, type q.

4. Go back to your home directory by typing

   cd

Make a new subdirectory by typing

mkdir tmp

Typing **ls** -F should show it listed as a directory. This directory will serve as a temporary (hence, the name, "tmp") directory. Now enter the new directory by typing

cd tmp

Use **pwd** to ensure that you are in /home/loginname/tmp. Type

cat /etc/rc.d/rc.sysinit

to print the file to the screen. Now do this again, but redirect the output to a file in your current directory instead of to the screen by typing

cat /etc/rc.d/rc.sysinit > testcat

Do a full listing of the tmp directory, **ls** -l, and record the result.

5. Try typing

   echo 'hi there'

   and see what happens. Now type

   echo 'hi there' > testecho

   then

   more testecho

   to redirect the output to the monitor. Now append more text to your testecho file:

   echo 'hi there' >> testecho

   then

   more testecho

Here are a few more QNX commands; these are useful for file manipulation.

cp *file1 file2*	makes a copy of file1 and names it file2
mv *file1 file2*	moves file1 to file2 (and deletes file1)
rm *file*	removes file

1.2.3 File Manipulation

1. Be sure you are in the "tmp" subdirectory of your home directory. Check this with **pwd**. Now make a copy of your "testcat" file by typing

   cp testcat testcat2

   Now move "testcat2" up one directory, to the parent directory, by typing

   mv testcat2 ../testcat2

   To make sure it got there, go up one directory

   cd ..

Then type **ls** -F. Now lets grab another file and copy it into the home directory. Use **pwd** to make sure you are in your home directory; if you are not there, use cd to get there. Now copy "testecho" into your "tmp" directory by typing

cp tmp/testecho testecho2

Note that the pathname to the "testecho" file is a relative pathname. Its full pathname would be "/home/loginname/tmp/testecho." You may test to see if the file is now in your home directory by using **ls**. To list all files that begin with the letters "test" you would simply type

ls test*

This is a feature provided by QNX which allows you to save typing. It is called a "wildcard." The **mv** command works in a similar way to the **cp** command. To erase a file, simply type **rm** *filename*.

2. What QNX commands would you use to accomplish the following

   a.) list all files in the current directory that start with a "q"

   b.) delete all files in the current directory that end in a number (0-9).

Thus far, we have been using the command line interface. QNX also has a graphical user interface (GUI), called "photon", which makes certain tasks easier.

1.2.4 Using the Graphical User Interface

1. To invoke photon, simply type **ph** at the command line prompt. Photon looks similar to many other GUIs in that it has a menu bar (which is located on the right hand side) and a large space in which other objects may be displayed. Spend a few minutes exploring photon. In particular

   a.) Click on the "terminal" icon. This opens what is called a "terminal." You should see the old familiar command line prompt. Type a few commands, such as **cd** or **ls** -F, just for fun.

   b.) Click on the "file manager" icon. This allows you to explore the directory tree which you previously explored using **cd** and **ls** -F.

   c.) Find your old file "testecho." Open it by right clicking on the file and selecting "open with..." and typing in "ped." This is the command to invoke the photon editor. Modify it, save it, and close it.

   d.) This is a bit of a tedious procedure to open a file; we would like to just double click on it. From the edit menu, select "associations." Enter "*" and then "ped" in the remaining fields. This will tell the file manager that when ever you double click on a file, it should invoke ped to open the file. Try to open "testecho" once again by double clicking on it.

2. When you are finished, click on the "Launch" button in the lower left corner of the GUI and select "shutdown" and "end photon session". This will end the photon session and you will return to the command line prompt, as you were before starting photon.

3. If the photon gui starts up every time you login in, and you do not want it to, you must create an empty file name "nophoton" in the /etc/config/system directory.

In order to start photon, you typed **ph** at the command line prompt. What you did was you ran an executable program called "ph" which allowed you to interact with the computer in a user friendly, albeit much slower, way. (You will probably find that the computer responds more quickly when you are using only the command line, *i.e.* when photon is not running.) Although photon is a fairly sophisticated program, it is not magical. Someone had to write this program. In fact, each of the commands we have been using, for example **ls** and **cd**, is itself a short executable program which allows the user to interact with the computer in a particular way. We will soon learn how to write and execute our own programs.

Your home directory contains a file that is executed when you log onto the system. (Most UNIX systems have a number of these so called "dot files" that do not appear when you do an ordinary **ls**. Typing **ls** -a forces them to reveal themselves, though.) The file named ".profile" contains personalized shortcuts and preferences and you can customize your machine a little by making changes to this file. Let's set up a few shortcuts.

1.2.5 Modifying the User .profile

1. Using either the command line or the graphical user interface, make a copy of the original .profile; for instance, from the command line, type

   cp .profile .profile.old

2. Invoke ped and add the following lines to your .profile.

   alias rm='rm -i' **alias ls**='ls -F' **alias** showprofile='more .profile'

   Make a personalized greeting and login message by adding something like

   echo 'Greetings, my friend!'

   to the end of the .profile. Save the file and close it.

3. Finally, logout and then log back in as before. Did the new file execute properly at login? Try typing showprofile and see if it works.

You should now be able to

- log on
- find your way around the computer's directory structure

- manipulate files (copying, moving, deleting, ...)
- use a minimal set of QNX commands, with simple redirection
- invoke Photon and edit and create text files

Take a moment to review these topics, then clean up the clutter you've created by deleting the various test files, both in your home directory and in the lab1 directory.

1.3 Using the Printer

You will want to print out some of the files that you generate so that you can put them into your notebook.[2] Printers are a shared resource which many computers have to use. Our computers are not connected by a network, so our sharing was a little primitive by modern standards.

1.3.1 Printer Setup

1. Upon booting, a program called an "enumerator" is run which detects what devices are attached to your computer so that you may use them. For example, the enumerator detects the mouse, the keyboard, the monitor, the hard drive, and the CD-ROM drive, and the printer. Hook up the printer to your computer's parallel port and reboot the system. You may need to check that in your BIOS Utility Setup, the parallel port is configured as "ECP" (for Enhanced Capabilities Port) rather than bi-directional."

1.3.2 Printing from the Command Line

1. To test whether the printer works, go to your home directory and type

   cat .profile > /dev/par1

   This sends the contents of the file ".profile" to the device located at parallel port 1. Your printer, which resides here, receives the data and prints the contents of the file.

[2]The instructions in this section applied to the case in which a printer was hooked up directly to the parallel port of the computer (*i.e.* the very old-fashioned way.) If you have a network card—which you should—then these instructions do not apply. You will need to do a bit of research to figure out how to print out your files so that you can include them in your lab notebook. You may need to jump ahead to Sec. 1.4.1 for notes on how to install and set up a network card. Then you can probably send files to another location from which you can print them. You should do this now so that you can print out the c-code you will be writing shortly so as to include it in your laboratory notebook.

1.3.3 Printing from the GUI

1. Your previous printout probably did not look so good. That is because the text was not formatted properly for the printer. In order to get pretty output, you need to run an intermediate program called a "filter" which formats the text for the particular printer which you are using. Invoke the GUI by typing **ph** at the command line. Go to your home directory and, using the photon editor, open your file ".profile." Select "print" from the photon editor menu. Notice that the name of the printer appears in a box. This is because it was properly detected by the enumerator program when the system was booted. When you print from the GUI, your printout should look much nicer because the photon editor automatically invokes the correct filters for you particular printer.

1.3.4 Reading Assignment 1

Read K&R pages 1 to 13.

1.4 Using the C Compiler

A program is a collection of commands written in a particular language which is designed to accomplish a particular task. We will be using a programming language called C, which was invented by researchers at AT&T Bell Labs. Unfortunately, C is not directly understandable by the computer; it must be "compiled." This is the process by which a "high-level" language, like C, is converted into "machine language," which the computer can understand. High-level programming languages look a bit like English; machine language looks like a bunch of ones and zeros.

In addition to being compiled, the program may be "linked" to other compiled programs which it requires in order to run properly. These programs are often located in a "standard library." We will discuss this more later. First we must install a C compiler that will allow us to compile the programs which we write. The particular compiler which we will use is called "qcc." This is short for "QNX C compiler."

1.4.1 Ethernet card setup

1. Be sure that your computer is turned off and unplugged.
2. Install a compatible ethernet card into one or your computer's PCI slots.
3. Plug in an ethernet cable. Then restart your computer, log in, and type the command

   ifconfig

   This should report if the ethernet card has been successfully detected: a line starting with *en0* should appear.

4. Now start up photon and check to see if your network settings are configured properly. Open the network configuration application. Under the "Device" tab, be sure to select DHCP and check the "enable device" check box. Then enter the correct domain name under the "Network" tab. You will probably use "wlc.edu". The IP address of the default gateway (at the time of this writing) is 172.17.4.4.

5. If all goes well, you should now be able to use QNX's web browser, "Voyager".

6. Use the **ifconfig** command line prompt to determine the IP address of your machine. Ask some of your classmates for their IP addresses. Then use the **ping** command to see if you can communicate with other machines on your local network.

7. You might also try to connect to your classmates' computers, which are running photon, using the **phditto** or the **phindows** command. You can explore how to use these (and many other) commands with the online qnx user guide.

1.4.2 FTP server setup

In order to exchange files between your home laptop or desktop computer and your qnx machine, you need to install **ftp** on your home machine. This should be available in an online software repository. In what follows, I will explain how to install *t*nftp on a machine running OSX. These commands will need to be altered if you are not using OSX; you may need to do some research to find the correct procedure.

1. First, install a package manager called *Homebrew* by first typing the following command at the command line in a terminal on your OSX machine

   /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

   Then, at the command line type

   brew install tnftp

2. Now go to your QNX machine and get the internet daemon (background process) running by typing

   intetd

   at the command line. While you are at it, double check the IP address of your QNX machine using **ifconfig**.

3. Now go back to your OSX machine and type

   ftp *IP address*

   When prompted, enter the username and password of your account on the QNX machine. You should get an *ftp*> prompt. At the prompt, type

get *testfile.txt*

to get the file named *testfile.txt* from your qnx into the current working directory of your OSX machine. Then you can type **exit** to kill the ftp program.

1.4.3 Installing the C Compiler

1. Photon has a program called an "installer" which makes it simple to install new software. Click on the "installer" icon on the menu bar. You now have two choices for installing software. The first is installing software from a www repository. For this you will need an ethernet connection. The second is installing from the QNX install disk. For this you will need to insert the QNX Realtime Platform disk into the CD-ROM drive. Follow the instructions to install the C/C++ development tools and libraries. After it is installed, you may remove the disk.

   You are now prepared to write, compile, link, and run your own C programs.

1.4.4 Compiling Your First C Program

1. Create a new subdirectory within your home directory. Call this directory "src". This is the directory in which you will be placing C source code. You may do this using either the command line interface or the file manager.

2. Create and save a new file, hello.c, containing the following text:

```
#include<stdio.h>

main()
{
    printf( "Hello, world!\n");
}
```

3. Open a terminal window and at the command line type:

   qcc hello.c

   This will compile your C program and generate an "executable" file called a.out. You may now try to run this executable file by typing

   a.out

   at the command line. Test to see if it works. You may get an error which states that a.out is "not found." This means that your PATH is set incorrectly.

To run a program, QNX must know where to look to find the executable file. When your computer is booted, a number of "environment variables" are initialized. Among them is a variable called "PATH," which contains a list of directories where programs, or "executable files," are typically located. If an executable is located in a directory other than one listed in the PATH, QNX will be unable to run the program.

1.4.5 Setting the PATH

1. At the command line, type

   env

   This will list all of the environment variables. Among them, you will see a variable called PATH, which contains a number of pathnames separated by colons. In which directories does QNX look for executables?

2. Although, the PATH environment variable is initialized at boot time, it can be modified at login time. Append the following line to your *.profile*.

   export PATH=$PATH:$HOME/bin:.

   This sets the PATH as follows

$PATH	the directories already in the PATH variable, and
$HOME/bin	the binary directory in your home directory, and
.	the present working directory

   If you do not have a "bin" directory, then create it in your home directory. This is where all of your executable (i.e. binary) files will be placed.

3. Logout then log back in and type **env** to be sure that the PATH environment variable has been set properly. Why must you logout and back in?

1.4.6 Running Your First C Program

1. Now you should be able to run your program by typing "a.out" at the command line. If it works, then you have successfully written, compiled, and executed a C program. Congratulations!

2. To give the executable file a more memorable name, you can use a "switch" when compiling the program. Try going to your "src" directory and then typing

   qcc hello.c -o ../bin/hello

   This will compile your C program and generate an executable file called "hello", located in your new "bin" directory. Make sure that a file called "hello*" is there. The asterisk indicates that it is an executable file. Do you understand why you typed the "../bin" before hello?

You may run your executable file by simply typing "hello" at the command line. Make a printout of your program, attach it to your notebook, and write down in your notebook what happens when you run it.

3. Make a copy of the hello.c program called hello2.c. Experiment on this copy by leaving out parts of the program, to see what error messages you get.

1.4.7 Fahrenheit-to-Celsius Conversion Program

1. Create a file (in your "src" directory) called f2c.c which contains the following program:

```
#include <stdio.h>

/*  print Fahrenheit-Celsius table
     for fahr = 0, 20, ..., 300 */
main()
{
    int fahr, celsius;
    int lower, upper, step;

    lower = 0;      /* lower limit of temperature table */
    upper = 300;    /* upper limit */
    step = 20;      /* step size */

    fahr = lower;
    while (fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + step;
    }
}
```

Compile the program, placing the executable in your "bin" directory. Run the program. Print out a copy of both the program and the output and attach them to your notebook. Describe what happens when you run the program.

2. Modify the program so that it prints a header above the table. The header should be justified so that it lines up with the numbers.

3. Now modify the program so that in addition to printing the Fahrenheit and Celsius temperatures, it also prints the temperature in Kelvin. Modify the header accordingly.

4. Change your program so that it uses floating-point arithmetic and prints out the results to one place behind the decimal point, right-justified.

1.5 Using the Floppy Drive

It is important to make backup copies of all the programs which you work so hard to create. There are several ways to do this. One way is to use the tar command. (Culture note: tar is an abbreviation for tape archive, and is typically used for backing up large quantities of data onto magnetic tape.) The computers we are using probably do not have a USB port. So we are going to use—yes—floppy disks! Welcome back to the 1990's!

1.5.1 Formatting and Initializing a Floppy Disk

1. First, we must get the floppy disk ready so that it can be used on our particular operating system. Insert it into the floppy drive. You will need to acquire superuser privileges. At the command line type

   su

   You will then be prompted to enter the root password.

2. We will use the command **fdformat** to format the floppy disk. Assuming you have a double-sided high-density disk, type

   fdformat /dev/fd0

   This formats the floppy disk, which QNX calls /dev/fd0 (**cd** to /dev and you will see fd0 there), to hold 1.44 Mbytes. If your disk is too small, you may need to specify the size, for instance,

   fdformat -s 720k /dev/fd0

3. Next, we must initialize the disk. This involves copying a few files to the disk so that QNX can figure out what to do with it. To initialize the disk, type

   /sbin/**dinit** -l diskname /dev/fd0

   You must include the full pathname since **dinit** is in the /sbin directory, which may be incompatible with your current PATH environment variable.

1.5.2 Changing User Privileges

1. Do a long listing of /dev/fd0 by typing

   ls -l /dev/fd0

   What does it show?

2. Only root has privileges to read from and write to the floppy drive. To change this, you will need root privileges. Make sure you do, then type

   chmod 777 /dev/fd0

   Do another long listing. What does it now show? You will now be able to use tar to archive your files on /dev/fd0 even if you are not the superuser.

3. Read about **chmod** using the HELP menu from within photon. Describe in your notebook what **chmod** 777 does. In particular, what do each of the "7"s denote?

1.5.3 Backing Up Files

1. Abandon your root privileges. Be sure you are in your home directory. Type

   tar -cvf /dev/fd0 .

   This means that the present working directory (.) is backed up to the floppy disk at /dev/fd0. (Notice the dot at the end of the command.) The v switch tells tar to be verbose, so that it tells us what it is doing with our files. You should see a list of files scrolling down the monitor. These are the names of the files tar is archiving.

2. What do the c and f switches mean?

3. Check to make sure that tar actually backed up all your files; type

   tar -tvf /dev/fd0

   Replacing c with t tells tar to read out the files it has saved.

1.5.4 Extracting Files from an Archive

1. Now we will convince ourselves that we can get back the files we saved by extracting them from the floppy disk. Create a new subdirectory named tmp off of your home directory and **cd** into that directory.

2. Type

   tar -xvf /dev/fd0

   Type **ls** to see what you created. Are all of your files there?

1.5.5 Mounting the Floppy Drive in the Filesystem

1. In the previous exercises, we tied up the entire floppy for the archive. In order to do a backup on the floppy disk and keep your files in a usual file format, we must mount the floppy in the usual file system. Obtain root privileges and then type

   mount /dev/fd0 /fd0

   This will mount floppy disk 0 as a directory /fd0. If things don't go well, you might wish to unmount and remount your floppy disk. Un-mounting the mount point /fd0 can be done by typing

   umount /fd0

   Aside: If you need to save files on a DOS formatted floppy, then you will need to use the -t flag when using the mount utility.

   mount -t dos /dev/fd0 /fd0

2. You will now probably need to reformat and re-initialize your floppy disk. This is because when we previously archived the working directory to the disk we overwrote the file formatting on the disk. Abandon your root privileges, be sure you are in your home directory, then archive your home directory by typing

   tar -cvf myfiles.tar .

   This puts the archive in a file called myfiles.tar. Now copy "myfiles.tar" onto your re-formatted floppy disk. The benefit is that we can still use the rest of the space on the floppy, for instance for other archives.

3. Do a **cd** /fd0 and verify that fd0 behaves like any other directory. Can you create a new subdirectory there? Can you copy other files thither?

1.5.6 Reading Assignment 2

Read K&R pages 13 to 34. Especially study Sects. 1.3, 1.4, 1.6, 1.7, 1.8, and 1.10.

1.6 Using a Graphing Program

There are a number of ways to graph data on the PC. We will use a stand-alone graphing program called Logger Pro. You should begin to familiarize yourself with how to use Logger Pro, if you are not already familiar with it. Later, we will use this to plot temperature data in order to measure the thermal conductivity of copper.

Chapter 2

Analog and Digital I/O

2.1 PC Architecture

The brains of the PC is an integrated circuit microprocessor. The microprocessor, called a central processing unit, or CPU for short, resides in a slot on the motherboard. The CPU processes chunks of data, called bytes. The CPU typically consists of an instruction decoder, which interprets instructions, an arithmetic unit, which performs operations such as adding or comparing data, a number or registers, which temporarily store data for processing, a program counter, which keeps track of the current location when executing a program, a cache memory, which holds data recently fetched from memory for quick access, and bus control circuitry, which handles communication with memory and I/O.

In addition to the CPU, the motherboard contains a number of other chips. These chips comprise what is called the chipset of the motherboard. If the CPU is the brain of PC, then the chipset is the central nervous system of the PC. It is the connection between the processor and everything else. It contains an interface between the CPU and the bus, memory controllers, bus controllers, I/O controllers, and more. The particular chipset with which your motherboard is endowed sets limitations on the processor, memory, I/O, and expansion capabilities of your motherboard.

Finally, the motherboard contains what is called the super I/O chip which integrates devices that used to be found on expansion cards in older systems. It might contain a floppy disk controller, a dual serial port controller, and a parallel port controller. Fig. 2.1 shows a schematic diagram of how the aforementioned three components fit into the architecture of one particular system.

Notice how, in Fig. 2.1, the North and South Bridge chips separate the system into three tiers. Communication within each tier proceeds *via* a set of shared lines, called a bus. Communication between the CPU, cache, and North bridge chip occurs on the Front Side bus; communication between the North and South bridge chips occurs on the PCI bus; communication south of the South Bridge chip occurs on the ISA bus. Using these three busses, communication is

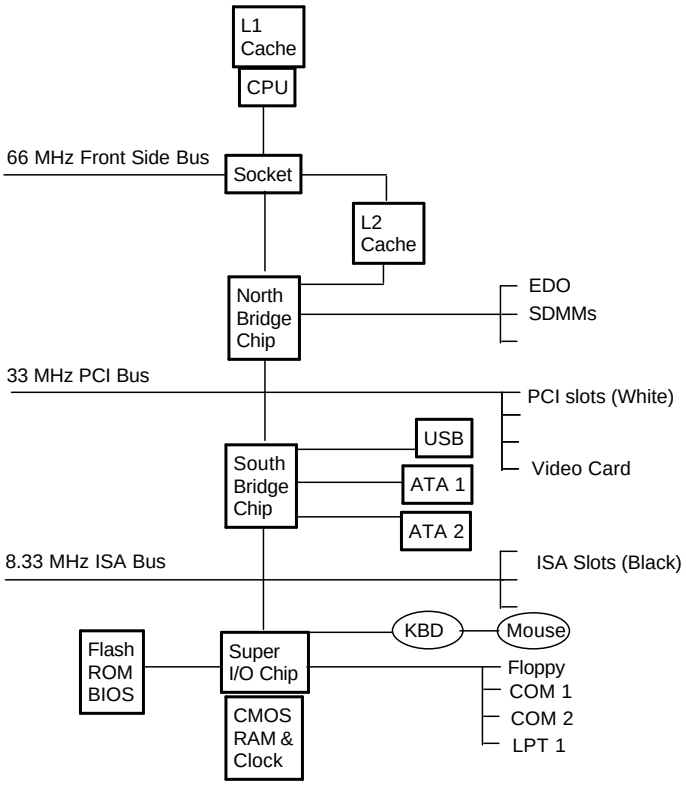

Figure 2.1: This is a typical Socket 7 (Pentium class) system architecture.

possible between the CPU and all of the peripheral devices.

Each bus consists of DATA lines, ADDRESS lines, and CONTROL lines. Each line can carry a high voltage corresponding to a digital one, or a low voltage corresponding to a digital zero. The front side bus for Pentium class processors have 64 DATA lines. Think of this like a 64 lane highway. The bus width and bus speed govern the bandwidth, or rate of data transfer on a particular bus. Thus, a 64 line DATA bus running at a clock speed of 66.6 MHz would have a bandwidth of 533 MB/second.

The ADDRESS lines are set to indicate which device is to send or receive the information carried on the DATA lines. The front side bus for Pentium class processors have 32 address lines; this allows the CPU to communicate with up to 4,294,967,296 distinct addresses. Think of the 32 ADDRESS lines as a telephone system with this many numbers which pass through the central switchboard: the microprocessor.

The CONTROL lines specify which action is going on. For instance, a signal on the read (RD) line indicates that data is being transferred into the CPU; a signal on the write (WR) line indicates that data is being transferred from the CPU; the IO/M wire indicates whether the address on the ADDRESS line should be interpreted as an I/O (port) address or a regular memory (Mem) address. The clock (CLK) wire controlled by the oscillator is the clock which determines how fast the data transfers will take place.

Communication between the computer and the outside world typically occurs in the following fashion. The microprocessor first places the binary representation of the location to be accessed on the ADDRESS wires. Then after waiting for the computer circuits to select the unique location to which this refers, it either sends or receives a byte of data on the DATA wires during the data cycle. The DATA wires may contain a binary representation of some number for manipulation or perhaps a machine language instruction. The chipset serves as a bridge between the various busses as well as between the busses and the computer's memory locations and I/O ports.

Computers usually have several types of memory; early computers had only Random Access Memory (RAM). RAM is essential for any computer since the fundamental principles of computer operation require Central Processing Unit (CPU) to repeatedly store and retrieve program instructions, data, and memory addresses. The term "Random Access Memory" means that it may be written to or read from in any order. A severe disadvantage of semiconductor RAM is that it does not remember anything after its power is turned off. Some computers have vital portions of their RAM protected by having a battery to provide the power in case of a power line failure.

Read Only Memory (ROM) has data stored in its memory cells at the time of manufacture which it retains permanently. It can be randomly accessed but that access is restricted to the read operation only. A ROM chip can be moved from one place to another without the data being lost as no power is needed to maintain data stored. There are several ROMS in the PC. One contains the Base System which is activated when the computer is turned on. Programs in the Base System initialize the computer and load the first programs from the

disk. Another ROM contains the information on how to form letters on the video screen.

2.2 Hardware Setup

2.2.1 A Look at the Motherboard

1. Be sure that the power to the PC is off and the cord is unplugged before opening up the chassis. Remove the screws that bind the outer case to the internal frame of the chassis and gently remove the case. Take a good look at the motherboard. See if you can identify the following components: the CPU, the BIOS ROM, the battery, the number and type of expansion slots, the motherboard power connector, the hard, floppy, and disk drive connectors, and the memory expansion slots. Make a labeled sketch of your motherboard in your lab notebook.

2. By looking at the chips, you can identify the chipset of your motherboard. Can you identify the north/south bridge chips. It might help to look in Ch 4, and specifically at Tables 4.6 and 4.7, in *Upgrading and Repairing PCs* by Scott Mueller. What limitations does your motherboard's chipset place on your system (e.g. what is the best processor your chipset supports?)

3. How might you "over-clock" your CPU?

As mentioned previously, the CPU communicates with peripheral devices *via* digital signals carried by the I/O (input/output) bus and expansion slots. Typical devices attached to the bus are a printer, a monitor, a hard drive, a sound card, or a graphics card. The expansion slots allow for a great deal of flexibility in your system; you may customize your system by inserting computer boards which suit your particular needs. For our laboratory experiments, we will be using a special device called an A/D ("A to D") board. This board has special capabilities which will be detailed in the following sections. Let us begin by installing the A/D board.

2.2.2 Installing the A/D Board

1. Take a good look at the A/D board. What is the model number? Read through the manual which accompanies the board, specifically sections 1, 3 and 8. These sections describe hardware installation and board specifications.

2. The jumper settings prescribe the base address of the board. You will need to know this so you can program the board, so record the settings in your notebook. It should be set to 0x300. How would you set the jumpers so that the base address was 0x400? Gently insert the A/D board into one of the unused ISA slots and screw it into the chassis. Consult the

instructor if it does not go in easily; it may not be lined up properly; do not force it! Plug in the computer and turn it on to make sure it works.

To make it convenient to communicate with our newly installed A/D board from the outside world, we will be using what is called a "protoboard", a strip of connectors which sits outside the chassis and allows us a great deal of flexibility in hooking up external circuits to the inputs and outputs of the A/D board. We should first set this up.

2.2.3 Setting Up the Protoboard

1. Obtain a flat gray ribbon cable with a 37-pin D type connectors on each end. Plug the connectors into the A/D board and the break-out board (which has screw-type terminals for connecting to your proto-board *via* jumper wires). Section 5 of the A/D board manual shows the pin-out for the 37 pin connector. If you have done it correctly, you should have connections like those in Tab. 2.1. Make a table in your notebook relating the DIP number and pin to the function of the wire. Be careful to get it right, as you will be referring to this table extensively in the future.

DIP	DIP pins	37-pin Connector
1	16-9	Analog Inputs 0-7
1	1-8	Analog GND and Digital Inputs 0-3
2	16-9	Digital Out 0-7
2	1-8	Digital Input 4-7 and 5V, 5V, -12V, 12V from PC Bus
3	14-12	Analog GND and Analog Out 0 and 1
3	1 and 2	Digital GND

Table 2.1: Protoboard pinout

2.3 Number Systems

In Sec. 1.4, we mentioned that although we write programs in high-level languages, such as C, which look a bit like English, the computer translates this into its own language, called machine code, which consists of only ones and zeros. A number system consisting of ones and zeros is called a binary number system. A hexadecimal number systems provide a shorthand notation for long binary numbers. It will be very useful to become acquainted with binary and hexadecimal number systems and their relationship to our decimal number system with which we are probably more accustomed.[1]

[1]The ancients used alternative numbering systems. For example, we inherited the sexagesimal (base-60) system of recording both time and small angles from the ancient Babylonians.[3] Thus, there are sixty minutes in an hour, sixty seconds in a minute, sixty thirds in a second, sixty fourths in a third, *etc.*.

The decimal number system is so called because it uses base 10. A number such as 293 can be described by the equation

$$293 = 2 \times 10^2 + 9 \times 10^1 + 3 \times 10^0 \tag{2.1}$$

That is: 2 one-hundreds, 9 tens, and 3 ones. The binary number system is so called because it uses base 2. A number such as 23 can be described by the equation

$$23 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \tag{2.2}$$

That is: 1 sixteen, 0 eights, 1 four, 1 two, and 1 one.

The hexadecimal number system is so called because it uses base 16. In Tab. 2.2 are shown correspondences between some binary, hexadecimal, and decimal numbers. Whereas the 'nice round numbers' in decimal are multiples of 10: $10^0 = 1$, $10^1 = 10$, $10^2 = 100$, and $10^3 = 1000$, the 'nice round numbers' in binary are multiples of 2: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, and $2^7 = 128$. Similarly, the 'nice round numbers' in hexadecimal are multiples of 16, such as $16^0 = 1$, $16^1 = 16$, $16^2 = 256$, and $16^3 = 4096$. Also, notice in Tab. 2.2 that every four binary digits corresponds to one hexadecimal digit. It is thus customary to group binary numbers in clusters of four digits. For example, the hexadecimal number 2B5 is written in binary as 0010 1011 0101. Often, to denote that a number is in hexadecimal notation, a 0x is place before it. In this notation, we would write the previous hex number as 0x2B5.

binary	hexadecimal	decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15
0001 0000	10	16

Table 2.2: Comparison of binary, hexadecimal, and decimal representation of numbers.

2.3.1 Number Systems

1. Write the following decimal numbers in binary and hexadecimal notation
   7, 42

2. Write the following hexadecimal numbers in decimal and binary notation
   0x03, 0xA7

3. Write out a binary multiplication table from binary 000 to binary 111. This should be an 8 by 8 table.

4. What type of logical operation(s) must be used in order to multiply binary numbers? (bitwise AND? OR? XOR? or something else?)

The A/D board is controlled and monitored by writing to and reading from eight consecutive 8-bit I/O addresses. The register map is detailed in section 6 of the A/D board manual. To write to a particular register, you simply send a number to the address of that register. The number you write to a register, when converted to binary notation, sets each of the eight bits in the register to either 1 or 0. The tricky part is deciding what numbers to send, and in what order, to accomplish the task which you want to accomplish.

2.4 Digital Output

In the following exercise, we will create a binary number display by writing hexadecimal numbers to the digital output register of the A/D board. You will assemble a circuit on your protoboard. It consists of 8 light emitting diodes (LEDs), each one connecting a port of the digital output register to digital ground. When the bits of the register are all set to zero, the ports are all grounded, and the LEDs are off. When a 1 is written to any bit of the register, the voltage at that port goes hi (+5 volts) and that LED turns on.

2.4.1 LED Binary Number Display

1. Hook the positive (shorter) terminal of an LED up to digital output port 0 and the negative (longer) terminal to ground. **Wait! Stop!** You must put a small resistor (about 200 ohms, perhaps) between the negative terminal of the LED and the ground. Otherwise, you will short out the digital output port of the A/D board and ruin your digital output ports!

   Do the same for each of the 8 digital output ports. Be sure to orient it correctly, since diodes can only pass current in one direction. Double-check that you have hooked up the diodes to the correct points on your proto-board and that they all go to the same ground. Make a sketch in your lab book of the circuit setup.

2. Save the following program. Then compile and run it. It writes a hexadecimal number to the digital output port. You will need root privileges to

run this program, as you will whenever you use the A/D board, so be sure to **su** before running the program.[2] Also, make sure to make a printout of the program and put it in your lab book.

```
#define BASE 0x300
#include<stdio.h>
#include<hw/inout.h>
#include<sys/neutrino.h>

main()
{
    unsigned char data;
    ThreadCtl(_NTO_TCTL_IO, 0);
    data = 0x01;
    out8(BASE+0x03, data);
}
```

3. What hex number would you assign to the variable data so that the first, third, and fifth ports are hi and the others are lo? Try it to see if it works.

4. The following program illustrates how binary counting works. Enter, compile, and run it. Also, place a printout in your lab book.

```
#define BASE 0x300
#include<stdio.h>
#include<hw/inout.h>
#include<sys/neutrino.h>

main()
{
    unsigned char data;
    ThreadCtl(_NTO_TCTL_IO, 0);
    for (data = 0x00; data < 0xFF; data = data + 0x01)
    {
        out8(BASE+0x03, data);
        in8(BASE+0x03);
        printf("%x\n",data);
        delay(250);
    }
}
```

5. What happens if you change the $<$ sign to a $<=$ sign? Why do you think this happens? (Hint: Consider the size of a variable of type `char`.)

[2] If this does not work, you can either do a little bit of research to find out what is wrong. Alternatively, you might just log in as root.

Let us go through this program in some detail. The first line of the program sets the base address of the board as 300 hex. The next three lines tell the program to include certain header (.h) files. When you installed the C compiler, you downloaded a number of libraries containing useful functions. We have previously used one of these functions: `printf()`. In order to use this function, the program had to know how the function works. The header file stdio.h, for standard input and output, contains a declaration for the `printf()` function. Our present program uses the `ThreadCtl()` and `out8()` functions. The function `ThreadCtl()`, declared in the header file sys/neutrino.h, gives the program special access to the input and output ports. The function `out8()`, declared in the header file hw/inout.h, writes data to a particular I/O port. Now consider the `main()` section of the program. After initializing a variable which is called data and is of type unsigned char, the function `ThreadCtl()` is run with some special arguments which allow the program to access the input and output ports. Next, we have a `for()`loop, which writes successively larger hex numbers to the output port using the function `out8()`. The `in8()` function updates the output ports. A delay of 250 milliseconds is inserted between each iteration of the for loop so that the numbers do not zip by too quickly.

2.4.2 Switching Speed

1. Write a program that flips an LED on and off once per second. Put a printout of the working program in your notebook.

2. Modify your program so that the switching speed is 500 Hz. Plug in the BNC cable with the oscilloscope probes into CH 1 of the oscilloscope. Hook up the probes across the diode and observe the diode voltage as a function of time. The ground lead of the oscilloscope probe should be connected to the ground of the system, *i.e.* the lower voltage of the LED. Never connect the oscilloscope probe ground to any point of a circuit which is not ground. Set the scope trigger control to AUTO; be sure the small switch on the probe tip is set to 1x; set the vertical scale to 1.0 V/DIV and the VARIABLE knob to the CALibrated position; set a 0 V baseline by using the ground switch and vertical position knob on the scope. Set the horizontal scale to 1 ms TIME/DIV. You may need to adjust the intensity, focus, position, and TRIG LEVEL to observe a nice square wave.

3. What is the highest frequency square wave that you can generate? Hint: You may want to just omit the delay statements in your program.

2.4.3 Reading Assignment 3

Read K&R Sects. 2.1, 2.2, 2.4, 2.5, 2.6 (pp. 42-43), 2.8, 2.10, 2.11, 7.1 to 7.5.

2.5 Analog to Digital Conversion

As mentioned in the previous section, computers communicate *via* digital signals-sequences of ones and zeros. On the contrary, most processes with which we are familiar are analog in nature. For example, the temperature outdoors changes continuously, not in discrete steps. In order for the computer to store or manipulate information about natural processes, it must convert analog information into digital information. This is accomplished by a device called an analog to digital converter, or ADC for short. The 12-bit AD574 analog to digital converter which is installed on the CIO-DAS08/JR-AO board can convert a voltage between -5 and +5 volts into $2^{12} = 4096$ parts. A "multiplexer" on the same board allows the user to select voltages coming from 8 different ports; the board is said to have 8 input channels.

Using the ADC is rather simple. First, a signal must be sent to the proper register to select the A/D channel and to initiate the conversion. Then, a short delay instruction is inserted to give the ADC time to perform the conversion. Finally, the register containing the data is read and interpreted. The following exercises gives you practice in writing a C program which performs an analog to digital conversion.

2.5.1 Using the ADC

1. Connect a potentiometer across a 5 volt power supply so that its two outer terminals are attached to ground and +5 volts. Observe the voltage of the wiper (the center connection on the potentiometer) using an oscilloscope. You should attach the oscilloscope ground probe to the ground of your power supply. (Remember, never connect the oscilloscope probe ground to any point of a circuit which is not grounded). When you turn the potentiometer back and forth, the voltage read on the oscilloscope should vary between 0 and 5 volts. Make a sketch in your notebook of the circuit diagram.

2. Create, and compile the following program. Make a printout and save it in your lab notebook.

```
#define BASE 0x300
#include<stdio.h>
#include<sys/neutrino.h>
#include<hw/inout.h>

main()
{
    unsigned int hibyte, lobyte, data;
    ThreadCtl(_NTO_TCTL_IO, 0);
    out8(BASE+0x02,0x00);
    for(;;)
```

```
        {
            out8(BASE+0x01,0x00);
            delay(1);
            hibyte= in8(BASE+0x01);
            lobyte= in8(BASE+0x00);
            hibyte = hibyte << 4;
            lobyte = lobyte >> 4;
            data = hibyte | lobyte;
            printf("%d\n",data);
            delay(500);
        }
    }
```

3. Connect the ground of the power supply to the ground of your DAQ card. Connect the wiper of the potentiometer to both the oscilloscope and ADC channel 0. Run the program you just wrote. Rotate the potentiometer shaft and note how the display on the scope and on the computer monitor change. If you hook up one end of the potentiometer to the + and -5 V terminals on the power supply, you should be able to obtain values between 0 and 4096 on your computer's monitor.

4. Modify the program so that the voltage is read on ADC channel 3. What change do you need to make?

5. Can you modify the program so it loops and checks for an end of conversion condition (EOC)?

Let's look at the `main()` section of the program. After `ThreadCtl()` is run, zero is written to the control register, `out8(BASE+0x02,0x00)`. This tells the A/D board that it should perform any of its A/D conversions on channel 0. The `while()` loop causes the instructions within the curly brackets to be executed repeatedly until CTRL-Z is struck. Consider the repeating section of the program. Writing zero to the A/D register, `out8(BASE+0x01,0x00)`, initiates an A/D conversion. Next, `delay(1)` causes a one millisecond delay. The next two lines read the two A/D registers, putting the values into the variables hibyte and lobyte. The board uses two 8-bit A/D registers to generate a 12-bit value. The next line assembles hibyte and lobyte into a variable called data which contains 12-bits of information. The << operator shifts the value of hibyte 4 bits to the left, filling vacated bits with zero. This is equivalent to multiplying it by 16. Likewise, the >> operator shifts the value of lobyte 4 bits to the right, filling vacated bits with zero. The | operator performs a bitwise inclusive OR operation on hibyte and lobyte. This generates a new 12 bit number in which hibyte comprises the 8 most significant bits and lobyte comprises the 4 least significant bits . Finally, data is printed to the screen and the loop is repeated after a 500 millisecond delay.

2.6 Resolution and Sampling Speed

Digital data is typically more robust to store than analog data. Hence, analog to digital converters are used quite often to store signals such as musical recordings or visual images. A high-fidelity digital recording should represent the true analog signal as faithfully as possible. This way, when we use a digital to analog converter (more on this later) to reproduce an analog signal from digitally stored data, it will (in the case of a sound recording) sound just like the original signal. Typically, some kind of "transducer" is used to first convert sound, color, or temperature into a voltage.

There are two main issues when performing analog to digital conversion: resolution and sampling speed. How closely a digital recording represents the analog input depends on the number of bits in the analog input register of the ADC. The number of bits is akin to the number of ticks on a ruler. The larger the number of ticks between zero and one inch, the higher the resolution with which the ruler may be read. For example, if a 12-bit ADC has in input range of 0 to 10 volts, the resolution is $10/2^{12} = 10/4096 = 0.0024$ volts. Changes in voltage which are smaller than 2.4 mV will not be detected by our 12 bit ADC.

2.6.1 ADC Dynamic Range

1. The resolution in voltage of an ADC is $\Delta V/2^n$ where ΔV is the total input range and n is the number of bits of the digital output. What is the resolution of an 8-bit ADC with input range +5 to -5 V?

2. Since the amplitude of an analog signal can be adjusted by an amplifier circuit to fill the input range of the ADC, the resolution can be better described by the dynamic range; this is the ratio of the maximum to the minimum voltage measurable by the ADC. The maximum is the ADC input range and the minimum is the resolution calculated above. What is the dynamic range of the above 8-bit ADC? The 12-bit one used in class? The ratio is usually expressed in decibels (dB), e.g., DR = 20 log (ratio) in dB. Give your answers in both forms, as a ratio and in dB.

The second complication arises when the analog signal is changing–perhaps rapidly. An ADC requires some "settling time" in which to perform the conversion. If the analog signal changes too quickly, the ADC will not be able to accurately track the changing signal. There is a famous result known as the Sampling Theorem, formulated by Shannon (1949), building on earlier work by Nyquist (1924), which states that in order to reconstruct the original signal from a sampled signal accurately, the ADC sample rate should be at least twice the highest frequency in the input signal.

For example, in order to do an accurate digital recording of musical sound with frequencies in the range 20-20,000 Hz, the sound must be sampled at least 40,000 times per second. This, 40 kHz, is called the Nyquist frequency. If there are frequencies of sound above 20 kHz which your ADC can detect (though your ear cannot), you must use a higher sampling frequency, even if your ear cannot

detect them. This is because the higher frequency components can masquerade as lower frequency components when sampled too slowly. This is an effect called *aliasing*. One way to guarantee that no higher frequency sound is being inadvertently sampled is to use a filter which cuts off any sound above 20 kHz from being recorded by the ADC.

2.6.2 Audio Digital Sampling

1. An eight channel digital recording studio wants to faithfully record the audio spectrum from 20 to 20,000 Hz. What must be the sample rate for each channel?

2. The studio wants to use a single multiplexed 16-bit ADC to digitize the signals (the ear is a sensitive detector); what is the maximum conversion time the ADC can have?

2.7 Functions

Functions in C are a way to simplify your programs. For example, `printf()` is a function which prints its argument (what is in parenthesis) to the screen. Though this may seem simple, it is actually a rather complex process. It is nice that we can just type a one line command and, bingo, characters appear on the monitor. Whenever we want to print something to the screen, we just invoke the `printf()` function. There are a large number of functions which are provided by the C libraries which you downloaded when you installed the C compiler. These greatly simplify the task of programming.

Now we will learn to write our own functions which suit our particular needs. Let us begin with an example. We will modify our ADC program from Exercise 2.5.2 so that the section of code in which the register-level programming is performed is placed in a separate function, called `adc()`.

2.7.1 Function Practice

1. Create, compile, and run the following program. Save a printout in your lab book.

```
#define BASE 0x300
#include<stdio.h>
#include<sys/neutrino.h>
#include<hw/inout.h>

unsigned adc( void );

main()
{
    unsigned int data;
```

```
        ThreadCtl(_NTO_TCTL_IO, 0);
        out8( BASE + 0x02, 0 );
        while(1)
        {
            data = adc();
            printf("%d\n", data);
        }
    }

        unsigned adc( void )
        {
        unsigned int hibyte, lobyte;
        out8(BASE+0x01,0x00);
        delay(1);
        hibyte= in8(BASE+0x01);
        lobyte= in8(BASE+0x00);
        hibyte = hibyte << 4;
        lobyte = lobyte >> 4;
        return( hibyte | lobyte );
        }
```

The first line below the #include statements is what is called the function declaration. Notice that the function `adc()`is of type unsigned. This means that when the function is called and it returns a value, that value is an unsigned integer. Also notice that the argument of the function `adc()` is of type void: `adc()`, unlike `printf()`, does not take an argument. Within the `while()` loop, the the function `adc()` is called. It performs the analog to digital conversion and then returns a value which is assigned to the variable called data. The variable data must be of the same type as the function `adc()`.

Notice that the function declaration comes before the main section of the program and that the function definition comes after the `main()`section of the program. Sometimes, the function declarations are collected and kept in a separate file called a header file. If a program uses a function, the name of the header file which contains its declaration must be placed in an include statement at the beginning of the program.

The `#define` statements are also usually placed in the header file. Furthermore, the function definitions are often collected and kept in a separate file. If a program uses a function, the compiled function must be linked to the program.

2.7.2 Building a Function Toolbox

1. Create three separate files: daqfuncs.h, which contains the function declaration; daqfuncs.c, which contains the function definition; and daq271.c, which contains the main section of the program. You should insert the line `#include daqfuncs.h` just below the other `#include` statements in

the .c file. You may need quotation marks around the header function name. This will include the function declaration in your program. You may also need the BASE definition in your daqfuncs.c file. Compile and link the code by typing **qcc** daq271.c daqfuncs.c -o ../bin/daq271

2. Thus far, we have been recording voltages as an integer between 0 and 4096 (decimal notation). We would like to convert this integer to a voltage. Write a function, called `itov()`, that converts the value recorded by the ADC into a voltage. Incorporate this function into your ADC program so that the voltage is printed to the monitor. Make a printout of each of your three files and place them into your lab notebook.

2.8 Arrays

Arrays in C are a way to store large amounts of data. For example, we may want to store a hundred voltages recorded using the ADC in an array called volt. The first voltage would be stored in the array element called volt[0]; the second in an array element called volt[1]; the last in an array element called volt[99]. Each member of an array is indexed by an integer. The first member has an index zero (in FORTRAN it is 1). The following exercises will give you practice in using arrays in C.

2.8.1 Array Practice

1. Enter, compile, and run the following program. It is a modification of the ADC program so that it takes a series of readings, stores them in an array, then prints the whole array to the screen after the last measurement is made. Do not forget to link the program to your function library when you compile it! Make a printout and place it into your lab book.

```
#define BASE 0x300
#include<stdio.h>
#include<sys/neutrino.h>
#include<hw/inout.h>
#include"daqfuncs.h"

main()
{
    int i;
    unsigned int data[10];

    ThreadCtl(_NTO_TCTL_IO, 0);
    out8( BASE + 0x02, 0 );
    for(i=0; i<10; i++)
        data[i] = adc();
    for(i=0; i<10; i++)
```

```
            printf("%lf\n", itov(data[i]));
    }
```

2.9 Pointers

Pointers are one of C's most useful and elegant features, but they can be a bit confusing until one gets used to them. A pointer is basically the address in memory of a variable. Let us give it the name px when the variable name is x. Most of the time we really do not want to know what the address is; it generally is a large integer which has no meaning to us. But it is useful to have ways of passing the location of a variable on to a function, for instance. The function can then go to that location and change the value which resides there. C has an operator which generates the address of x. It is the & sign: if we program $px = \&x$, then px will contain the location of x. There is another operator *, which is essentially the inverse of &. Thus, $*px$ is the contents of the location identified by px, which is of course nothing other than x. So we see that $*px$ is equal to $*(\&x)$, and this is equal to x. & and * are examples of unary operators (they operate on one object). Other unary operators which we have encountered are the incrementation and decrementation operators ++ and $--$.

Pointers must be declared as to the type of variable to which they are pointing: for instance double $*px$; if the variable is a `double` or `long` $*px$; it is a long integer. One of the useful aspects of pointers is that one can perform arithmetic operations on them. One simply treats them as integers. Thus $px + 1$ points to the location which is next in line beyond px. But here the type to which px is pointing must be considered. If px points to a character, then $px + 1$ points to the next byte beyond px; but if for instance px points to a double, $px + 1$ points eight bytes (the size of a double) beyond px. The value of px can be modified, for instance by the operation $px = px + 1$ or, equivalently, $px++$. You can see how pointer arithmetic can be used to access successive memory locations. The use of pointers is strongly encouraged in the C programming language because it encourages the user to think in terms of how the computer stores and accesses memory locations.

Interesting and useful is the relation between pointers and array names. The name of an array is just like a pointer to the first element of the array, except that one cannot modify it by arithmetical operations (the name must stay what it was defined to be). Nonetheless, if you defined an array $a[N]$, then $*(a + 5)$ for instance is the same as $a[5]$. You can also assign the value of a to a pointer, say pa, and then increment pa and thusly proceed through the elements of the array. As an example, say you write $pa = a$. Then you do some arithmetic on pa which, say, is equivalent to $pa = pa + 7$. Now $*pa$ is equal to $a[7]$. Here you have to remember that successive members of an array are in successive memory locations.

2.9.1 Pointer Practice

1. Rewrite your ADC program from Sec.2.8.1 so that it uses pointers rather than arrays. When you get it working, make a printout and save it in your lab book.

2.10 Memory Allocation

In the previous exercises, we have declared arrays of a given size. If we use pointers to move through the relevant memory locations, this seems rather primitive! We should be able to just say that we want to reserve memory for a certain number of members of a specified variable type. We can do this with the `malloc()` function. The function `malloc()` will reserve space in memory for what we need, and it will tell us where that space starts. A typical statement would be

```
pa = (int *) malloc( 15 *sizeof(int));
```

Here *pa* is a pointer to the first byte of the array to be stored, `(int *)` shows that `malloc()` returns a pointer to an integer array, 15 is the number of integers we want to store, and `sizeof(int)` simply returns the machine-dependent number of bytes occupied by an integer. Very simple, no? No! But you will get used to it. Of course, better programming form would be to not have the number 15 in the code; one might consider replacing it by `DIM`, and to have a define statement `#define DIM 15` at the beginning of the program.

2.10.1 Malloc Practice

1. How many bytes does the `malloc()` statement shown above allocate for your PC? How many would it allocate if the variable were a double?

2. Rewrite your ADC program of Sec.2.9.1 without an explicit declaration of an array, using `malloc()` instead. Make a printout and save it in your lab book.

2.11 Making and Retrieving Data Files

Here we will learn about how to read and write data files. This is obviously important in any experiment. The libraries associated with C provide several functions which are useful for interacting with files. Among them are `fopen()`, `fclose()`, and `fprintf()`. As the name implies, `fopen()` "opens" a file, i.e. gets it ready for data to be read from it or written to it. More precisely, it creates a FILE structure which is a collection of information about a file; but fortunately it is not necessary for us to worry about the details of how it works unless we would like to. The syntax for `fopen()` is

```
fp=fopen("filename","mode")
```

Here, "filename" is a character string equal to the name of the file. Typically it comes from the standard input (i.e. you type it on your keyboard after you are prompted) and it is read by the program using `scanf()` with %s. Alternatively, the filename might be generated in the program. In that case, it would typically be written on a string using `sprintf()`. `sprintf()` works pretty much like `printf()` except that you have to specify the name of the string on which you want to write: for instance

```
sprintf(adc_file, "n_%d", day);
```

would yield the filename n_10 if day=10. This name would reside in the string called adc_file. adc_file should have been declared as a character, or as a character array if it is to hold long filenames. The "mode" is "r," "w," or "a," depending on whether you want to read from, write on, or append to a file. If you use "w," the writing begins at the beginning, and whatever else may have been there will be gone! If all went well, fp is a pointer to a FILE structure which must have been declared by the statement `FILE *fp` near the beginning of the program. If for some reason the file could not be opened (because perhaps some other process was using it), `fopen()` returns 0. Thus, good programming practice would be to test for this. Once the file has been opened, `fprintf()` is easy to use; it is the same as `printf()`, except that its first argument must be the file pointer fp of the file to which we want to write and which we obtained using `fopen()`. So here is an example:

```
    fprintf(fp, %d %lf \n", i, volt);
```

Once we have done the writing, we should close the file with `fclose(fp)` Closing the file does more than you might think! Much of the data which you write to the file are actually kept in a buffer (some special fast memory set aside for this) until enough has accumulated to warrant actually writing it on the disk (this is a slow operation to initiate, and it is more efficient to do it in larger chunks rather than a few bytes at a time). If your computer should happen to "crash" for one reason or another, the data in the buffer would be lost; `fclose()` flushes out the buffer, and makes sure that all your precious results are safely stashed away on the hard disk. Finally, here is an example of a section of a program:

```
 FILE *fp; ... while((fp=fopen(adc_file,"w"))==0) {
    printf("file busy\n");
    sleep(1);
} fprintf(fp,"%d %lf\n",i, volt);
fclose(fp);
```

It tries to open the file once a second until it succeeds. Don't forget: the second and further times you open the file to add to it, you better use "a"!

2.11.1 Saving Data to a File

1. Modify your ADC program so that it saves the data to a file. Open the file and make sure that the data was stored properly. When you get it working, make a printout of the program and save it in your lab book.

We have used `printf()` and its relatives `sprintf()` and `fprintf()` to write data on files. Once we save data to a file, we may want to read the data from the file. Reading from standard input is achieved with `scanf()`. It is a little trickier than `printf()`, and if we do not use a little care, we will get nonsense. A typical `scanf()` statement might look like

```
scanf("%d %lf",&i, &v);
```

Here, the things in quotes give the type of the variable to be read from the input and put into the soft memory of your computer. In the example, `i` must have been declared an integer (%d) and `v` must have been declared a double (%lf). The function `scanf()` has to know this because it needs to know how many bytes to write to define the desired information which makes up the variable of interest. The remaining function arguments are the addresses of the variables. This tells `scanf()` where to put the information which makes up the variable. So don't forget the & !!! unless you are using a pointer inside `scanf()`.

2.11.2 Reading from the Standard Input

1. Modify your ADC program so that it asks the user for an input channel, then selects the correct channel on which to perform the conversion. You will need to use a line like

   ```
   scanf("%i",&ch)
   ```

   Here, ch is a variable of type integer. Make a printout of your program and save it in your lab book.

If you want to read from a file, you use `fscanf()`, and the only difference is that the first argument is the file pointer (which we called fp above). So to read from the file adc_file which we used earlier (assuming that it has not been closed with `fclose()`), we would program

```
fscanf(fp,"%d %lf", &i, &volt);
```

if `i` had been declared as an integer and volt had been declared as a double.

2.11.3 Reading Data from a File

1. Write a program that reads the data stored in the adc_file, places the values in an array, and prints them to the screen. Make a printout of the program and save it in your lab book.

2.12 Digital to Analog Conversion

The inverse of analog to digital conversion is, you guessed it, digital to analog conversion. Your compact disk and DVD players use digital to analog conversion

to convert digitally stored data into music or your favorite movie. Our A/D board provides two channels of analog output using an AD7237 digital to analog converter which is located on the board. Each channel may be programmed to provide an analog output ranging from -5 and +5 volts with 12-bit resolution by writing the correct sequence of numbers to the correct registers on the A/D board.

Digital to analog conversion is fairly straightforward. First, a number corresponding to the desired voltage is sent to one of the analog output registers. Next, the analog output channels are updated by reading the digital I/O control register. In the following exercise, we will write a simple program that sends an integer to the DAC.

2.12.1 Keyboard Input to DAC

1. What output voltages (and on what channels) would result by sending the following binary integers to the following analog output registers?

case	BASE+0x04	BASE+0x05	BASE+0x06	BASE+0x07
1	1101 0001	0000 1000	0000 0000	0000 0000
2	0000 0000	0000 0000	1000 0001	1000 0100
3	1101 0001	0000 1000	1000 0001	1000 0100

2. Enter, compile, and run the following program. Monitor the analog output channel using the oscilloscope to be sure that the program is working sensibly.

```
#define BASE 0x300
#define RESOLUTION 4096
#define RANGE 10.0
#include<stdio.h>
#include<sys/neutrino.h>
#include<hw/inout.h>
#include<stdlib.h>

int vtoi(double);
unsigned getbits(unsigned, int, int);

main()
{
    unsigned int ch, data;
    double v;

    ThreadCtl(_NTO_TCTL_IO, 0);
    printf("Enter voltage:\t");
    scanf("%lf",&v);
```

```
    data=vtoi(v);
        out8(BASE+0x04,getbits(data,7,8));
        out8(BASE+0x05,getbits(data,11,4));
        in8(BASE+0x03);
}

int vtoi(double v)
{
        return( (RESOLUTION-1) * ( v+5. ) / RANGE );
}

unsigned getbits( unsigned x, int p, int n )
{
    return( x >> (p+1-n)) & ~(~0 << n );
}
```

The most difficult part of this program is understanding the `getbits()` function. What is it doing? Why do we need two `out8()` commands?

3. Clean things up a bit by collecting the two new functions `vtoi()` and `getbits()` into your daqfuncs.c function library. Create a new function, `dac()`, which contains the register level functions for digital to analog conversion. Also, modify the program so the user can specify which analog output channel is desired. Make sure your new program works, then print it and place it in your notebook along with a printout of your (growing) "daqfuncs.c" and "daqfuncs.h" files.

2.13 Digital Signal Processing

2.13.1 Recording and Playback

1. Use a function generator to apply a sinusoidally varying voltage to one of the analog input channels of your A/D board. Use a variation of your ADC program to sample the voltage over a reasonably long time interval. Save the data in an appropriately named file

2. Use a variation of your DAC program to generate a voltage which is read from your stored data file. Use the oscilloscope to monitor the analog output channel of your A/D board. You may wish to have a loop which continually cycles through the stored data. You may also need to truncate the data file so that it goes through an integer number of complete cycles of the sine wave. This will make the trace look nicer on the oscilloscope.

3. What is the highest frequency sine wave which you can accurately sample and playback?

4. Attempt some digital signal processing by applying some of the following operations between recording and playback.

a.) inversion

b.) halving

c.) absolute value

d.) derivative (using finite differences)

e.) anything else

2.13.2 Lissajous Figures

1. As in the previous exercise, sample a sinusoidal wave from the function generator. Then sample another sinusoidal wave of a different frequency. Store these in separate data files.

2. Playback one wave on one DAC channel and the other on the other DAC channel. Hook these up to channels 1 and 2 of the oscilloscope. Set the MODE of the oscilloscope to 'X-Y' and look at the pretty Lissajous figures. How, exactly, this is done, depends upon the make and model of your oscilloscope. The 'X-Y' mode creates a parametric plot (remember this from your math classes?) of $y(t)$ versus $x(t)$, where t serves as the independent parameter on which x and y depend.

Chapter 3

Thermistor Experiments

3.1 Introduction

We have been learning a great deal about QNX, computer architecture, the C programming language, and analog and digital input and output. Let us now take what we have learned and build a temperature controller. We will later use our temperature controller to maintain a sample of copper at a known temperature so we can measure how its thermal conductivity depends upon its temperature. The first step in building our temperature controller will be to build a heater.

3.2 Power, Specific Heat, and Thermal Response Time

NOTE: when any wiring is done or changed be sure to turn off the power supply. Even though the wiring is very simple, it is still a good idea to become accustomed to using wire color codes to help you. Red is used for positive power supply connections, green for ground, and the standard electronic color code if there is an easy correspondence to data line numbers. Taking the time to do this will make it easier to trace the circuit to find errors when something doesn't function correctly. In addition it is much easier to find test points when a scope or other test instrument are to be used.

3.2.1 Setting up the Heater Circuit

1. Set up the circuit as shown in Fig.3.1. The heater is a segment of manganin wire wrapped around a block of aluminum. Manganin, like tungsten in an incandescent light bulb, has rather high electrical resistivity compared to copper. When the pushbutton is depressed, the circuit is completed and current flows through the heater. Be sure to make a sketch of the circuit in your laboratory notebook.

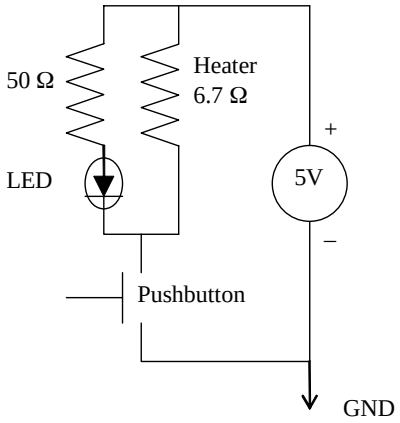

Figure 3.1: Basic heater circuit; the LED indicates when the heater is on.

2. The mercury thermometer should be inserted into the aluminum block. Press the pushbutton and observe the temperature rise. While the button is pressed, record how fast the temperature rises (roughly, in degrees per second).

3. Does the temperature immediately begin to fall when you release the pushbutton? Why do you think it behaves in this manner?

4. Determine how fast the temperature falls when the heater has been turned off (roughly, again, in degrees per second).

Indeed, you might have noticed that there is a lag between the heat supply and the thermometer response. The power supplied by the heater wire is given by

$$P = \frac{V^2}{R_{heater}}. \tag{3.1}$$

In the time interval Δt, an amount of heat energy $\Delta Q = P\Delta t$ is supplied to the aluminum block. The relationship between heat capacity, C, heat input, ΔQ, and temperature change, ΔT, is given by the formula

$$\Delta Q = C\Delta T. \tag{3.2}$$

Since the heat capacity is just the mass of the object, m, times its specific heat, c, the temperature rise per second is given by the formula

$$\frac{\Delta T}{\Delta t} = \frac{P}{mc}. \tag{3.3}$$

3.2.2 Calculating the thermal response

1. Using the above formulas, calculate the rate of temperature rise expected for your aluminum block. You will need to estimate the volume of your aluminum block. According to the *CRC Handbook of Chemistry and Physics*, the specific heat of aluminum is about 0.22 cal/g-C and the specific gravity (the ratio of the weight to that of water at 4 degrees Celsius) of aluminum is about 2.7. You should look up these values and record in your lab book what page this information can be found on.

2. Does your calculation match your experimental results from the previous exercise? Why might it not match? What do you think are the sources of error in your theoretical estimate?

You may have found that, based on the mass and specific heat of the aluminum block, the calculated thermal response time is shorter than the observed thermal response time. This is because the thermometer itself takes time to respond to changes in the block temperature.

This is in fact a general characteristic of measurement instruments: they take time to respond to a signal. The thermometer is called a first-order instrument; its response is determined by the differential equation

$$\tau \frac{dT_{therm}}{dt} + T_{therm} = KT_{block} \tag{3.4}$$

K is called the static sensitivity (or the calibration factor). Usually it is equal to one so that when the thermometer temperature, T_{therm},stops changing (such that the first term equals zero), T_{therm} equals the block temperature, T_{block}. τ is a characteristic thermal response time of the thermometer. It depends on the heat capacity of the thermometer and the quality of the thermal contact between the block and the thermometer. For an abrupt change in the block temperature, the solution to this differential equation[1] is given by

$$T_{therm} = KT_{block}\left(1 - e^{-t/\tau}\right) \tag{3.5}$$

Notice that when $t = \tau$, the temperature will have risen to $1 - e^{-1} \approx 2/3$ of its final value. So introducing a sudden step in the block temperature and observing the response of the thermometer provides a way of determining the time constant of the thermometer. If, instead of a step change in the block temperature, the block is continually heated, one may expect a perpetual lag in the thermometer temperature reading behind the actual temperature of the block.

3.3 Thermistor basics

Unfortunately, a mercury thermometer does not readily lend itself to computer automated temperature measurement and control. Instead, we will use a thermistor. This is a device whose electrical resistance depends strongly upon temperature.

3.3.1 Setting up the thermistor circuit

1. A thermistor should be mounted in the aluminum block which you used in the previous exercise. It will serve as our thermometer. Using the 5 volt power supply as V_0, assemble the circuit as shown in Fig. 3.2 on your protoboard. Make a sketch of the circuit in your laboratory notebook.

2. The relationship between V_T and V_0 is given by the voltage divider formula

$$\frac{V_T}{V_0} = \frac{R_T}{R_T + R_1} \tag{3.6}$$

[1] The thermal problem is formally analogous to the electrical problem in which a capacitor is being charged through a resistor by a power supply. The heat capacity is analogous to capacitance; the thermal resistance between the heater and the aluminum block is analogous to resistance.

Therefore, by measuring V_T, you can determine the thermistor resistance. Invert the above equation to obtain a formula for R_T in terms of the other variables.

3. Write a program which periodically (once per second) reads the voltage, V_T, on ADC CH0 over the course of, say, 180 seconds, converts these values into thermistor resistance values, and prints them to the screen.

How does the resistance of a thermistor depend upon temperature? According to the classical *Drude theory*, the resistivity of a metal depends upon the number density of charge carriers, n, the charge of each charge carrier, e, and the mass of each charge carrier, m, as well as on the time, τ, between collisions of the mobile charge carriers and the stationary protons.

$$\rho = \frac{m}{e^2 n \tau} \tag{3.7}$$

Typically, each atomic nuclei donates one or more electrons to the pool of mobile electrons. The number density of electrons is therefore huge, perhaps $10^{22}/\text{cm}^3$, and is nearly independent of temperature. The collision time, however, decreases as the temperature rises, since the protons vibrate more vigorously as the temperature rises. Therefore the resistivity of metals rise with temperature.

On the other hand, the resistivity of a homogeneous semiconductor, such as Germanium, drops when the temperature rises. Although the Drude theory is not entirely applicable to semiconductors (actually, neither is it to metals), it gives qualitative insight into their behavior. For semiconductors, the number density of charge carriers is very small at low temperatures. This is because the electrons are typically tightly bound to the atomic nuclei. As the temperature rises, atomic vibrations become increasingly energetic. The energy required to strip an electron from a nuclei is given by

$$E_g = k_B T_0, \tag{3.8}$$

where k_B is Boltzmann's constant and T_0 is some characteristic temperature. The probability of an electron being liberated from any given atom by thermal agitation is given by

$$\begin{aligned} P &= e^{-E_g/k_B T} \\ &= e^{-T_0/T}. \end{aligned} \tag{3.9}$$

Thus, the number density (number per unit volume) of free electrons in a semiconductor varies as

$$n = n_0 e^{-T_0/T}. \tag{3.10}$$

Using the Drude theory, we see that the resistance of a semiconductor may be written as

$$R = R_0 e^{T_0/T}. \tag{3.11}$$

In this expression, R_0 is a minimum resistance value at very high temperature, T_0 is an activation temperature (in Kelvin), and T is the absolute temperature

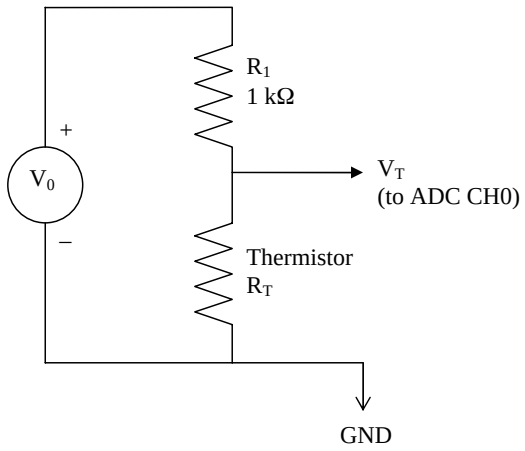

Figure 3.2: Thermistor circuit

(in Kelvin) (0 degrees Celsius = 273.16 Kelvin). The exponential variation of the resistance with temperature arises due to the rapid variation of the number of charge carriers with temperature. R_0, on the other hand, depends on the dimensions of the semiconductor and some factors which depend only weakly upon temperature.

3.4 Feedback and control

Although we have not yet calibrated our thermistor against a known temperature scale (we will do that a bit later, using the mercury thermometer), we may still use it as a rudimentary thermometer since we know that its resistance varies with temperature. In this section, we will continuously monitor the temperature of the aluminum block by measuring the resistance of the thermistor, and turn the heater on or off so as to maintain a predetermined thermistor resistance (and hence temperature).

Although the output ports of your A/D board have a range of several volts, they cannot supply much electrical current and so, in general, cannot drive external circuitry loads directly. HEXFETs are one variety of enhanced mode power FETs (Field Effect Transistors) which are particularly suited for controlling large amounts of power by using the digital signals coming out of a computer.

3.4.1 Controlling the HEXFET

1. Set up the circuit as shown in Fig. 3.3. It will act like the push button switch you used earlier but will be controlled by the computer. When a HI signal is applied to the gate of a HEXFET, the device conducts current like a closed switch; when a LO signal is applied, the device acts like an open switch, i.e., it has infinite resistance. Make a sketch of the circuit in your lab book.

   NOTE: you may need to use two separate power supplies, instead of a single one, as shown in Fig. 3.3. This is because: if the HEXFET is drawing a lot of current,you may inadvertantly load down the power supply, causing its voltage to drop. This, in turn, will change your thermistor reading.

2. Connect the gate of the HEXFET to digital output port 0 but wait to turn on the power to the power supply. Use the following program to test the control of the HEXFET. Turn on the power supply to the HEXFET circuit only after the program is running so that the HEXFET is not inadvertently left on for a long period of time and the thermometer is overheated. Make a printout of the program and put it in your lab book. What does the program do?

```
#define BASE 0x300
    #include<stdio.h>
```

```
#include<hw/inout.h>
#include<sys/neutrino.h>

main()
{
    int i;
    ThreadCtl(_NTO_TCTL_IO, 0);
    for(i=0;i<30; i++)
    {
        out8(BASE+0x03, 0x01);
        delay(1000);
    }
    for(i=0;i<30; i++)
    {
        out8(BASE+0x03, 0x00);
        delay(1000);
    }
    for(i=0;i<30; i++)
    {
        out8(BASE+0x03, 0x01);
        delay(1000);
    }
    out8(BASE+0x03, 0x00);
}
```

3.4.2 Temperature regulation

1. Now that you can turn the heater on and off, write a program to regulate the temperature. The program should first ask the user to enter a target thermistor resistance. Be careful to pick a reasonable value for the target resistance. Next, it should execute a loop in which it 1) reads the voltage across the thermistor, converts this to a resistance, and prints its value to the screen, and 2) turns the heater on or off so as to approach the target thermistor resistance. Run the program and demonstrate to your laboratory instructor that the thermistor stabilizes at the target resistance. When testing, be sure to turn off the heater manually (or with a statement at the end of your program) to ensure that the heater does not overheat.

3.5 Thermistor temperature calibration

We now have a working temperature controller. Unfortunately, we do not know the temperature at which we are controlling. We will now calibrate the thermistor against a mercury thermometer inserted into the aluminum block. Re-

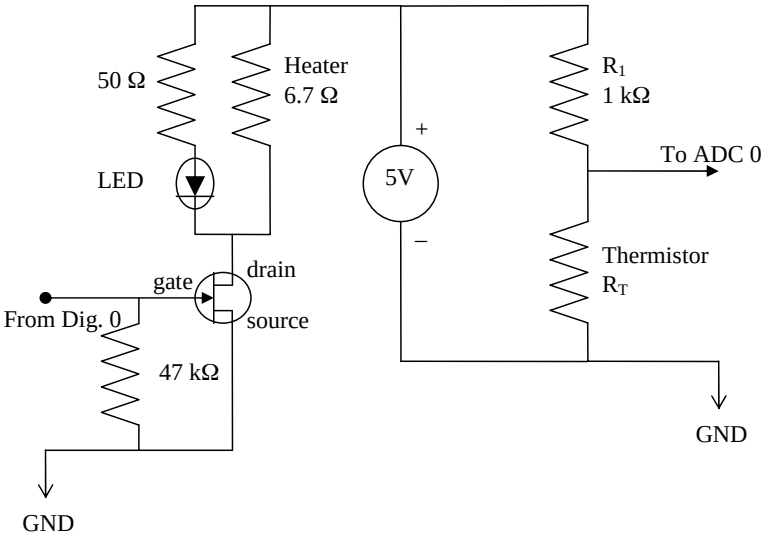

Figure 3.3: HEXFET temperature controller

member that the block may take a short while to equilibrate at the target temperature. What we want to do, then, is to select a target resistance, initiate our temperature controller, wait a short while until the resistance stabilizes, then record the resistance and the corresponding temperature from the mercury thermometer. We will repeat this procedure for several values or resistance between room temperature and about 100 degrees Celsius. This will give us calibration data: thermistor resistance versus temperature.

3.5.1 Thermistor calibration runs

1. Modify your previous program so that, in addition to printing the resistance values to the screen, it saves them in an array. It should collect data once per second for perhaps 240 seconds. After data collection, it should prompt the user to enter the temperature shown on the mercury thermometer and should record this with a `scanf()` statement. Finally, it should create an appropriately named (e.g. date_kelvin) data file with two columns. In the first column, it should record the time; in the second column, it should record the thermistor resistance value. Test to be confident that the program you have written generates a data file. When you get the program working, make a printout and put it in your lab book.

2. Inspect your data file (preferably by using a plotting program) to be sure that the resistance had equilibrated by the end of the data collection run. If it had, then you know the resistance corresponding to one particular temperature. If it had not, then you should modify your program so that it collects data for a bit longer. You may even be able to get away with collecting data for a bit less time. Once you are satisfied, perform 10 or 15 calibration runs over a wide range of temperatures. Make a table of resistance versus temperature in your lab notebook.

3.6 Least squares fitting to data

Thus far, we have a table of our thermistor's resistance and a number of temperature values. We would like to find the resistance at *any* given temperature within the range in which we calibrated our thermometer. To do so, we will need to find a mathematical equation which fits out data: Eq. 3.11. This equation can be cast in the form of a straight line; then we can use a linear least-squares fit to fit this formula to our data. Let us be a little more precise. Taking the natural logarithm of Eq. 3.11 gives

$$\ln R = \ln R_0 + \frac{T_0}{T}. \tag{3.12}$$

By setting

$$y = \ln R \qquad A = T_0 \qquad x = \frac{1}{T} \text{ and} \qquad B = \ln R_0, \tag{3.13}$$

Eq. 3.12 becomes

$$y = Ax + B, \tag{3.14}$$

which is the equation for a straight line. By finding values for A and B from the linear plot, values for R_0 and T_0 can be easily calculated. In doing the experiment, you have acquired data at a sequence of values of temperature T_i or alternatively $X_i = 1/T_i$. Each of these temperatures yielded an experimental resistance value R_i or alternatively Y_i. The model equation yields a theoretical resistance value R_i^{th} for each temperature, *i.e.*, for each X_i a theoretical value $Y_i^{th} = \ln R_i^{th}$ is given. The task is to find values for A and B to minimize the error between the experimental and theoretical values, $E_i = Y_i^{th} - Y_i$. A common type of analysis minimizes the sum of the squares of the individual errors. Calling the total square of the error E_T, we get

$$\begin{aligned} E_t &= \sum_i E_i^2 \\ &= \sum_i \left(Y_i^{th} - Y_i\right)^2 \\ &= \sum_i \left(AX_i + B - Y_i\right)^2 \end{aligned} \tag{3.15}$$

To minimize this error with respect to the parameters A and B we take derivatives with respect to A and B and set them equal to zero:

$$\begin{aligned} \partial E_t/\partial A &= 0 \\ &= \sum_i 2X_i \left(AX_i + B - Y_i\right) \\ \partial E_t/\partial B &= 0 \\ &= \sum_i \left(AX_i + B - Y_i\right) \end{aligned} \tag{3.16}$$

Taking A and B out of the summations and collecting terms gives

$$\begin{aligned} AS_{XX} + BS_X &= S_{XY} \\ AS_X + BS &= S_Y \end{aligned} \tag{3.17}$$

where

$$\begin{aligned} S_{XX} &= \sum_i X_i^2 \\ S_Y &= \sum_i Y_i \\ S_X &= \sum_i X_i \\ S &= \sum_i 1 \\ S_{XY} &= \sum_i X_i Y_i \end{aligned} \tag{3.18}$$

Then solving for A and B

$$\begin{aligned} D &= SS_{XX} - S_x^2 \\ A &= \frac{SS_{XY} - S_X S_Y}{D} \\ B &= \frac{S_{XX} S_Y - S_{XY} S_X}{D} \end{aligned} \tag{3.19}$$

3.6.1 Least squares fit to data

1. Write a program to find values for A and B using a linear least squares fit to arrays of data $x[i]$ and $y[i]$. Use the program to obtain the model fit to your resistance and temperature data and obtain values for T_0 and R_0. When you are satisfied with your program, make a printout and put it in your lab book.

2. Using your favorite plotting program, plot the theoretical fit as a line together with your experimental values as open circles. Make a printout and put in in your lab book.

The least squares fit assumes the measured data will be randomly scattered about the theoretical fit. The plot of the previous exercise does not show this clearly. A quick visual test of this assumption is to make a plot of the difference between the data and the fit, *i.e.*, plot the errors E_i. These are called the *residuals*.

3.6.2 Plot of residuals

1. Make a plot of the difference between the measured data and the theoretical fit to the data of the previous exercise. By inspection determine if the assumption of random errors was justified. Print out your plot and put it in your lab book.

3.6.3 A better temperature controller

1. Now that we have determined R_0 and T_0, we may easily convert any temperature into a thermistor resistance and vice versa. Modify your temperature controller program so that instead of asking for a target resistance, it asks for a target temperature. It should then proceed, as before, to regulate the temperature at the target value for some time. Note: In order for your c-code to invoke mathematical functions, such as sin() or ln(), you will need to include the "math.h" header file. You may also need to use the flag "-lm" to link to the math library when you compile your code.

2. Finally, change your program so that instead of using the digital output port to turn the heater on or off, it uses the analog output port. Instead

of simply turning the heater on or off, which causes significant overshoot, the DAC should apply a voltage to the gate of the HEXFET which is proportional to the difference between the target temperature and the measured temperature. This should allow for more precise temperature control. To display how well your temperature controller works, collect a data set and make a plot of temperature versus time. How well can you control the temperature? One way to characterize the level of control is the root mean square temperature fluctuation.

3.7 Errors in data and parameters

In fitting a theoretical model to data in the least squares method, the implicit assumption has been made that each data point has been measured with the same reliability. This is often not the case and it is then important to include a measure of the data reliability when fitting a model to these data. Another result of frequent interest which is not obtainable by the simple least squares fit is to determine how much the fitted parameters can vary without straining the fit to the data (how good is the fit?).

To make a statement of how good a measurement is we usually quote the value measured together with an expected error; for example a voltage is $V \pm \Delta V$ volts. An accepted definition of ΔV is that it is the root mean square (rms) value of the random error inherent in the measurement.

Consider a plot of a proposed theoretical fit and the data points $Y_i \pm e_i$ at a series of parameter values X_i. Assume the X_i are well determined (no uncertainty). The true variation of $Y(X)$ is given as some function of X. For sake of discussion assume that Y is of the form $Y(X) = AX + B$ where the parameters A and B are to be determined.

The total error can now be written as

$$E_T = \sum_i \left[\frac{(AX_i - B) - Y_i^{ex}}{e_i} \right]^2 \tag{3.20}$$

where e_i is the error in the data point Y_i. A small error e_i at data point Y_i will cause the difference between the model and the data point to be weighted heavily in the sum. Thus the points with small errors have a stronger effect on the fit. Proceeding as before yields the same formula for A and B except that

now

$$\begin{aligned} S_{XX} &= \sum_i X_i^2/e_i^2 \\ S_Y &= \sum_i Y_i^2/e_i^2 \\ S_X &= \sum_i X_i/e_i^2 \\ S &= \sum_i 1/e_i^2 \\ S_{XY} &= \sum_i X_i Y_i^{ex}/e_i^2 \end{aligned} \tag{3.21}$$

By standard rules of error propagation analysis, the errors in the estimates of A and B are determined to be

$$\begin{aligned} e_A^2 &= S/D \\ e_B^2 &= S_{XX}/D \end{aligned} \tag{3.22}$$

where $D = SS_{XX} - S_X^2$ as before.

Keep in mind that the estimation of the parameters $A \pm e_A$ and $B \pm e_B$ by the least squares method is a statistical one, i.e., given the data and the model junction, the calculated parameters A and B are the most likely ones for the system. The method assumes that the errors made in the measurements are random. It does not consider any systematic errors which may be lurking in your data. These last need to be ferreted out by careful thought and experimentation.

3.7.1 Errors in thermistor data

1. Make an evaluation of the error in your resistance determinations with the ADC and reanalyze the thermistor data with error considerations. To simplify error analysis, assume some reasonable constant error ($\Delta R_i = \Delta R$ for all i) and simplify the error equations by factoring the errors out of the sums.

3.7.2 Scientific Writing Assignment

An important part of this course is learning how to write a scientific paper. To this end, you should write a scientific paper, no more than four pages in length, which describes your work in calibrating your thermistor. This paper will serve as a warm-up, so to speak, for your final paper in this course. It will provide you with an opportunity to obtain feedback from your instructor regarding your scientific writing. To help you, your instructor will provide you with a few well-crafted scientific papers that can serve as examples. Your paper should include the following elements:

1. A *heading* which includes the title, the name of the authors, and the affiliation of the authors (institution and address).

2. An *Abstract* which states succinctly the most important results of your experiment.

3. An *Introduction*, which provides background on the problem which is being addressed in the paper. The introduction typically describes previous theoretical or experimental work that has been done on the problem.

4. An *Experimental apparatus and procedure* section description of your apparatus design and operation. This is the place to report the dimensions of your apparatus, the make and model of any equipment used, the experimental conditions, and the sequence of events that were carried out to perform a typical experiment.

5. A *Results and discussion* section in which you present your data, data plots, and your method of analysis including any formulae. Are your results reasonable? What are the most significant sources of error in your experimental results? This section may also involve some discussion for difficulties or avenues for further work.

Chapter 4

Thermal Diffusion Experiments

4.1 Introduction

In the previous chapter, we built a temperature controller that employed a heater and a thermistor. A temperature scale for the thermistor was obtained by calibrating it against a mercury thermometer. We will now put our knowledge to use to study the properties of matter. In particular, we will apply a heat pulse to the end of a copper rod and measure the temperature at points along its length. From these measurements, we will determine the thermal conductivity and the heat capacity of our sample of copper.

4.2 Heat flow equation

When we calibrated our thermistor, we modelled the temperature dependence of its resistance using a particular formula, $R = R_0 e^{T/T_0}$. We did not choose this formula arbitrarily; rather it was physically motivated by our understanding of the conduction of electrons in a semi-conducting material. Furthermore, the parameters that appear in the formula, T_0 and R_0, have physical meanings: the activation temperature, and the limiting resistance at very high temperatures, respectively.

We will use a similar procedure in modelling heat flow in our copper rod. We will use what is called the diffusion equation. The specific heat, c, and the thermal conductivity, k, are parameters that appear in the diffusion equation. In this section, we will derive the diffusion equation.

Nearly all energy eventually becomes heat. Heat is a form of energy associated with the random motion of particles. Furthermore, heat tends to flow from warmer to colder objects. As an example, if a warm object, T_a is brought into thermal contact with a cold object, T_b, an amount of heat ΔQ will flow from

a to b. The temperature change of object a is given by $\Delta T_a = -\Delta Q/m_a c_a$, that of object b is given by $\Delta T_b = -\Delta Q/m_b c_b$. Here, m is the mass of the object, measured in kilograms, and c is its specific heat, measured in Joules per kilogram-Kelvin.

The rate of heat transport is called the power, and is given by

$$P = \Delta Q/\Delta t. \tag{4.1}$$

Instead of two objects, we might consider heat transport through a rod of length L whose left end is held at a warmer temperature than its right end. The amount of heat deposited in a particular segment of length Δz, in time Δt, is simply the difference between the amount of heat coming in from the left and going out to the right:

$$\Delta Q = [P(z) - P(z + \Delta z)]\Delta t. \tag{4.2}$$

Linearizing this equation, we obtain

$$\Delta Q = \left[P(z) - P(z) - \left(\frac{\mathrm{d}P}{\mathrm{d}z}\Delta z\right)\right]\Delta t. \tag{4.3}$$

Thus, the rate of heat flow is given by

$$\frac{\Delta Q}{\Delta t} = -\left(\frac{\mathrm{d}P}{\mathrm{d}z}\right)\Delta z. \tag{4.4}$$

Now, recall that when a bit of heat ΔQ is added to a mass m, its temperature increases by the amount $\Delta T = \Delta Q/mc$. The rate of temperature rise for a segment of length Δz and density ρ can then be written as

$$\frac{\Delta T}{\Delta t} = \frac{1}{\rho A c \Delta z}\frac{\Delta Q}{\Delta t}. \tag{4.5}$$

Combining Eqs.4.4 and 4.5, we see that

$$\frac{\Delta T}{\Delta t} = -\frac{1}{\rho A c}\left(\frac{\Delta P}{\Delta z}\right). \tag{4.6}$$

In seems reasonable that the power, or rate of heat transport, is proportional to the gradient of the temperature, dT/dz, as follows:

$$P = -kA\frac{\mathrm{d}T}{\mathrm{d}z}. \tag{4.7}$$

Here, A is the rod's cross sectional area (square meters) and k is the thermal conductivity (watts/cm-kelvin). Plugging this into the previous equation, we obtain

$$\frac{\Delta T}{\Delta t} = -\frac{1}{\rho A c}\left(\frac{\mathrm{d}}{\mathrm{d}z}\left[-kA\frac{\mathrm{d}T}{\mathrm{d}z}\right]\right). \tag{4.8}$$

If the thermal conductivity is independent of temperature, and hence position, which we will assume, it can be pulled outside of the spatial derivative along with the cross sectional area. In the limit of $\Delta t \to 0$, we can then write Eq. 4.8 in the form:

$$\frac{dT}{dt} = \frac{k}{\rho c}\frac{\mathrm{d}^2 T}{\mathrm{d}z^2}. \tag{4.9}$$

This is the *diffusion equation*. We may define $D \equiv k/\rho c$, the *thermal diffusivity*, whose units are cm^2/second. For our copper rod, $k \approx 4 \times 10^7$ erg/cm-s-K, $\rho \approx 9$ g/cm^3, and $c \approx 4 \times 10^6$ erg/g-K. (We here use the CGS, rather than the MKS system of units.) You should look up the precise values for these physical quantities in the *CRC Handbook of Chemistry and Physics*.

The diffusion equation may be solved analytically by considering the ideal case of an infinitely long one-dimensional rod and assuming that all of the heat is deposited at one location at $t = 0$. We will consider this case. The solution to the diffusion equation is then

$$T' = B_1 + B_2 \frac{1}{\sqrt{t}} e^{-z^2/4Dt}. \tag{4.10}$$

The reason for the T-prime notation will become apparent in a moment. (It does *not* indicate a derivative.)

4.2.1 Diffusion equation solution

1. Determine the thermal diffusivity for a piece of copper
2. Verify that the above solution in fact satisfies the diffusion equation.

We have here two unidentified parameters, B_1 and B_2. After a long time has passed, the second term becomes zero, and the rod should be at ambient temperature, T'_0. Thus, $B_1 = T'_0$. Defining $T = T' - T'_0$, we may write the solution to the diffusion equation as

$$T = B_2 \frac{1}{\sqrt{t}} e^{-z^2/4Dt}. \tag{4.11}$$

T is therefore the difference between the ambient temperature and the temperature of the rod at a particular point at a particular time. We will call it the "excess temperature." To identify B_2, we recognize that the total heat added to the rod, Q, may be found by integrating the quantity $dq = mcT$ over the length of the rod:

$$\begin{aligned} Q &= \int_0^\infty \mathrm{d}q \\ &= \int_0^\infty \mathrm{d}z\, \rho A c T \\ &= \frac{\rho A c B_2}{\sqrt{t}} \int_0^\infty \mathrm{d}z\, e^{-z^2/4Dt}. \end{aligned} \tag{4.12}$$

Performing this Gaussian integral and solving for yields

$$B_2 = \frac{Q}{A\sqrt{\pi\rho ck}}. \tag{4.13}$$

4.2.2 Integration practice

1. Perform the above integral and verify the equation for B_2

Now we can write the solution to the diffusion equation in terms of meaningful variables:

$$T = \frac{Q}{A\sqrt{\pi\rho ckt}} e^{-z^2/4Dt}. \tag{4.14}$$

This equation for T is still a bit complex; to elucidate its meaning, let us identify a characteristic time scale, τ, and a characteristic temperature scale, Θ, associated with a particular position along the rod. We define

$$\begin{aligned} \tau &= \frac{z^2}{4D} \text{ and} \\ \Theta &= \frac{2Q}{A\rho cz\sqrt{\pi}}. \end{aligned} \tag{4.15}$$

Now, we can write the solution to the diffusion equation in dimensionless form:

$$\frac{T}{\Theta} = \sqrt{\frac{\tau}{t}} e^{-\tau/t} \tag{4.16}$$

4.2.3 Reduced temperature and time

1. If one joule (1×10^7 erg) of heat is added to a 1/8 inch diameter copper rod, what is the value of Θ at a the location of each of your thermistors?
2. What is the value of τ at the location of each of your thermistors?
3. Verify that, using the above definitions, you can write the solution to the diffusion equation in this dimensionless form.
4. Verify that the maximum of T/Θ occurs at a time t such that $t/\tau = 2$. This will be used in your experiments to determine the diffusivity of copper.
5. Verify that at $t/\tau = 2$, $T/\Theta = 0.43$. This will be used in your experiments to determine the value of ρc.

What is the point of this kind of analysis? If we have a thermometer placed along the rod a few centimeters from the heat source, we can measure its temperature as a function of time. If we know the amount of heat added to the rod, Q and the maximum temperature, T_{max}, at the location of the thermometer, z, we can then determine the value of ρc. Furthermore, from the time at which T_{max} occurs, we can determine τ, and hence D. Finally, from D and ρc, we can determine k, the thermal conductivity of copper

4.2.4 Heat capacity

1. Verify that

$$\rho c = \frac{0.86Q}{\sqrt{\pi}AzT_{max}} \tag{4.17}$$

4.3 Experimental setup

The apparatus which we will use to measure the thermal conductivity of copper consists of a 1/8 inch outer diameter (o.d.) copper rod soldered to an aluminum base. Affixed to the rod with Stycast 2850 (thermally conductive) epoxy are (i) a 2.2 Ohm carbon resistor at one end which serves as the heater, and (ii) two thermistors spaced along the length of the rod. Insulated 10 mil[1] o.d. manganin wire is used to make electrical connections between the heater and thermistors and an 8 pin hermetic electrical feed-through epoxied into the aluminum base. Insulated 22 gage copper wires connect the feed-through to the proto-board.

The proto-board contains two circuits, very similar to the ones you have constructed previously for your temperature controller. On the left side of Fig. 4.1 is the heater part of the circuit. Notice that now we use an 18 volt power supply. We want to dissipate a lot of energy in the heater in a short time period, so you should use a power supply that can deliver up to four or five amps.

On the right side of Fig. 4.1 is the thermometer part of the circuit. The only difference between this and your previous circuit is that now we will be reading the voltage across two thermistors at different locations on the copper rod.

4.3.1 Circuit assembly

1. Assemble the circuit on your proto-board, as shown in Fig. 4.1. Make a sketch of the circuit in your lab notebook, clearly labelling the circuit elements.

4.3.2 Control program

1. Write a program that will, first, turn the heater on for an interval of one or two seconds; second, record the temperature of the two thermistors several times per second and store them in an array; third, ask the user for an output file name; fourth, open an output file and store the temperature versus time data in the data file. Print out a copy of the program once you get it working and place a copy into your lab book.

[1] One "mil" is one thousandth of an inch. This is sometimes also called a "thou".

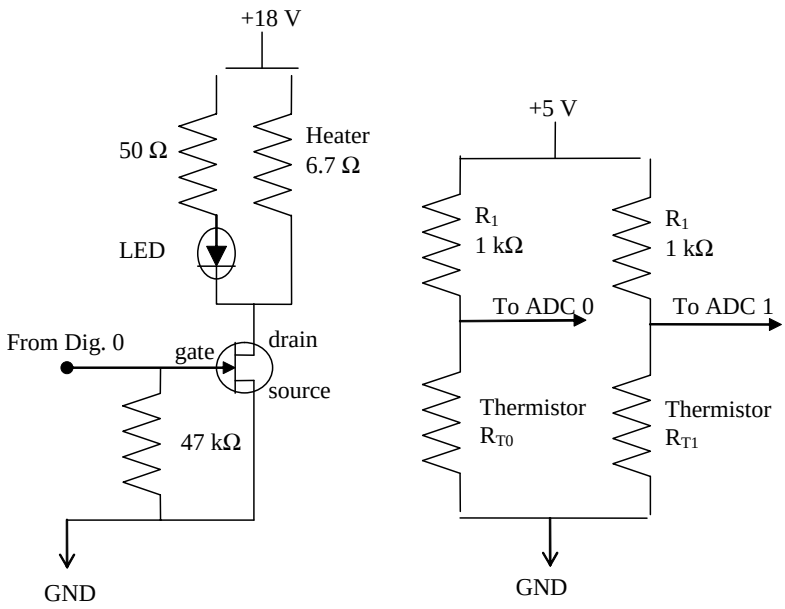

Figure 4.1: The heater and temperature control circuits for the thermal diffusion experiments.

4.4 Conducting the experiment

4.4.1 Data collection

1. Use your control program to apply a short heat pulse to the end of the copper rod and measure the evolution of the temperature at the thermistor locations as a function of time. You will need to collect data for perhaps four minutes in order for the temperature to return to room temperature after applying the heat pulse.

4.4.2 Data analysis

1. You will need to use a plotting program, such as *Logger Pro* or *Igor*, to analyze your data. If you do not have a plotting program on your PC, and you do not have a network connection, then you will need to save your data to a floppy disk. You may need to save your files on a DOS formatted floppy. First, you will need to mount your floppy drive. Type

   mount -t dos /dev/fd0 /fd0

   Now, cd to the directory in which your data is stored. To recursively copy all of the data files, type:

   cp -R * /fd0

   Now you can retrieve the data from this disk using a machine running Microsoft Windows, if necessary.

2. View your data using a plotting program. By identifying the the maximum temperature for a particular thermometer, you can determine ρc, the product of the density and the specific heat of copper. By identifying the time at which this maximum occurs, you can determine the value of the diffusion constant, D. From these two measurements, you can determine the thermal conductivity, k.

3. You may need to make the following correction. Recall that the analysis assumed that the heat was imparted to the rod instantaneously. In our case, we dissipated energy over perhaps one second. Therefore, you may need to shift the right (on the time axis) by a half a second to (roughly) account for this.

4. Also, you may obtain better results for the thermal conductivity if you perform a curve fit of the solution to the diffusion equation,

$$T = \frac{Q}{A\sqrt{\pi\rho c k t}} e^{-z^2/4Dt}. \tag{4.18}$$

   to your data, allowing ρc and D to be fitting parameters.

5. Another source of error arises from the fact that not all of the heat Q dissipated in the heater travels through the copper rod; some of it travels

through the surrounding air. Would this lead you to overestimate or to underestimate the thermal conductivity of the copper rod?

6. Evacuate the air from the vicinity of the copper rod and repeat the experiment to obtain a more accurate value for k.

4.4.3 Final paper

Now that you have set up your experiment, collected your data, and performed some analysis, you will need to share your results with the world! To this end, you will need to write a short scientific paper. The paper should be no more than four pages long, and should be composed in the style of the Physical Review. You may wish to look at a few papers to get a feel for the formatting. Your paper should include the following elements:

1. An *abstract* which states succinctly what you did and what you found.

2. An *introduction or overview* which describes the relevance of your experiments and previous work on this subject. You will need to do a literature search to find previous work on the topic. I'd start with the SAO/NASA Astrophysics Data System (http://adsabs.harvard.edu) or a Google scholar search.

3. An *experimental apparatus and procedure* section description of your apparatus design and operation. This is the place to report the dimensions of your apparatus, the make and model of any equipment used, the experimental conditions, and the sequence of events that were carried out to perform a typical experiment.

4. A *results and discussion* section in which you present your data, your method of analysis and your results. Be sure to compare your results with any previous or accepted values. If they differ, you need to provide a reasonable explanation as to why this is so. Which results are most reliable? What are the most significant sources of error in your experimental results?

Appendix A

Scientific Linux

Scientific Linux is a free Linux distribution which was developed at Fermilab and CERN and released in 2004. What follows is a brief guide to (i) installing and setting up Scientific Linux, and (ii) installing the USB drivers that will allow you to use the USB-6008 data acquisition unit from National Instruments (rather than the CIO-DAS08-JR ISA board from Measurement Computing.) This is for students who are taking the alternative path of using Scientific Linux rather than QNX Realtime operating system. In essence, the course will be the same either way. You will be carrying out a set of exercises aimed at teaching you to set up a computer to automate data acquisition in the laboratory. The difference is that the course manual has been written with the QNX operating system in mind. This means if you choose to use Linux, you will need to adapt the exercises accordingly.

System Administration

First, you will need to obtain a disk image from the Scientific Linux distribution website. If you have a 32 bit machine, make sure you have the appropriate disk image from the downloads page. Also, be sure that the disk image fits on an available CD or DVD. There is a 487M disk image called

SL-64-i386-2013-04-17-LiveMiniCD.iso

for Scientific Linux 6.4. Burn it (I used Disk Utility on a Mac and a superdrive to burn a CD). Then install it on your target machine by inserting the CD and booting your computer.

Follow the instructions in the Scientific Linux installer, which should begin right away. When you are asked to set the root password, let us agree to set it to *phy215*. And let us agree to name our computers phy215-SL1, phy215-SL2, and so on. After installation, try to reboot. If you get a panic, try to boot again. It should work.

When setting up your user account, choose whatever username and password you find appropriate. If you have an ethernet card inserted, then you can connect to the ethernet. This will be handy when configuring your computer. Speaking of configuration, you should go through the initial config exercises described at *server-world*. Here is the *url*:

`www.server-world.info/en/note?os=Scientific_Linux_6&p=download`

Do some of the exercises behind the initial config tab. Personally, I skipped the FW and SELinux, Configure network and Configure services exercises, but you can take a look at these. I did a system update with **yum**, a scientific linux command which allows you to install and update software packages.

yum -y install rpm forge -release

You may not be able to do this unless you have root privileges. So try to switch user from yourself to root.

su root

You will be prompted to enter the root password. Then type

visudo

Enter your password. This will allow you to edit the sudoers file. (Normally youd use the vi text editor, but you cannot use this to edit sudoers. Add the following line at the end.

yourusername ALL = (ALL) ALL

This will give yourusername the ability to act as system administrator. Test this by trying to reboot the compute without using sudo.

/sbin/shutdown -r now

Now try this

sudo /sbin/shutdown -r now

You will be asked for a password and then the computer will reboot.

Now, go through all of the exercises in the PHY215 lab manual, to the best of your ability using Scientific Linux. When you get to the section on installing a c compiler, you will need to use yum again.

sudo yum install gcc

You may also wish to update the software on your computer. Use

sudo yum update

Now try writing some c code, just like in the lab manual. To compile the code and run it, you may need to export the correct path to the PATH environment variable. The PATH can be checked by typing

env

Analog and digital input and output

If you are running Scientific Linux on a desktop which lacks an ISA slot, you will not be able to use the CIO-DAS08-JR/AO board, as your classmates are doing. Instead, you will be using a USB-6008 data acquisition unit. In order to use this device, you will need to download and install some drivers, rebuild your kernel, and do a firmware update on the USB device. This is describe below.

First, go the the National Instruments website. Ind the drivers for data acquisition. By poking around, you should find a file called

nidaqmxbase-3.7.0.iso

Download this file into your home directory perhaps. Go to your home directory and create some subdirectories.

mkdir DAQmx tmp

Now mount your iso file.

sudo mount -o loop nidaqmxbase-3.7.0.iso tmp

Recursively copy the files into tmp.

cp -r no-preserve=ownership tmp/* DAQmx

Unmount your mount point.

sudo umount tmp

Remove it.

rm -fr tmp

And go into your DAQmx directory.

cd DAQmx

You should find there an INSTALL file. Run it

sudo ./INSTALL

You may run into a problem with kernel support. So you will need to execute the following command

sudo yum install kernel_devel

That will install the appropriate kernel support. Reboot your computer and try to reinstall the DAQ drivers. Next, plug in your USB-6008 device and see if your computer recognizes it by running *lsdaq*

/usr/local/bin/lsdaq

You may need to update the firmware on the device. So execute *FWUpdate*

/usr/local/natinst/nidaqmxbase/bin/FWUpdate

Now try *lsdaq* again. It should find the usb device.

Next exercises:

Your next task is to try to reproduce the digital and analog input and output activities in the PHY215 lab manual. I have not done these, but there are a number of exercises included in the directories which were installed with nidaqmxbase. These illustrate how to use c code to perform analog and digital input and output with the USB-6008 device. There is also a data logger GUI which comes installed with nidaqmxbase, which you may wish to try out. Notably, it lacks the ability to output analog signals, so it cannot do everything you need it to do. There is a lot of information available on the National Instruments website which may be of use to you in writing c code. Good luck!

Bibliography

[1] P Horowitz and W Hill. The Art of Electronics, 1989.

[2] B Kernighan and D M Ritchie. The C programming Language, 2017.

[3] Daniel F Mansfield and N J Wildberger. Plimpton 322 is Babylonian exact sexagesimal trigonometry. *Historia Mathematica*, 44(4):395–419, 2017.

[4] S Mueller. Upgrading and Repairing PCs, 2003.

[5] D E Simon. An Embedded Software Primer, 1999.

[6] B G Thompson and A F Kuckes. *IBM-PC in the Laboratory*. Cambridge University Press, Cambridge, 2009.