Mobility, Ergonomics & Multitasking in Text Entry

A Senior Thesis in User Interface Design

Krispin Leydon

Advisor: Ted Cooley

Dartmouth College

Spring 2000

Table of Contents

Appendices

Mobility, Ergonomics & Multitasking in Text Entry

A Thesis Investigation in User Interface Design

Krispin Leydon
Winter '00
ENGS 87, Dartmouth College
Advisor: Ted Cooley

Abstract

Of all the human interface devices (HIDs) commonly found with a personal computer, the keyboard has changed the least. Its design remains nearly identical to the QWERTY keyboards people have used since the development of the first production typewriter in 1874 (2, 32).

The proliferation of tiny portable and embedded computer systems, the explosion of repetitive stress injuries (RSIs) corelated to keyboard use, and the value placed on mobility and the ability to multitask in the emerging knowledge based "virtual" workplace are creating a stronger demand for portable, user-friendly, flexible and ergonomic HIDs than has ever existed.

A number of alternative interfaces vie to replace the ubiquitous QWERTY design, including optimized key layouts, ergonomic keyboards, voice & handwriting recognition software, body-motion capture devices, and "chording" keyboards: keyboards that require fewer keys, but require simultaneous key strokes to identify a character. Direct mind control of computers through nerve impulses is becoming a reality, but this technology belongs more to the future than to the present (6). Each of these alternatives has advantages over standard keyboards, however no combination of advantages has been strong enough to challenge the inertia held by standard keyboards. No solution currently available on the mass market intregrates ease of use, speed, portability, flexibility and ergonomic design comprehensively in one design.

Over the course of the past two terms, I've conducted a thesis investigation in the field of man/machine interface design. The primary goals of this investigation were:

1) To obtain an introduction to human factors and usability engineering.
2)To refine an understanding of the specific interface design problem I cared about adressing.
3)To design a sound *process* by which an effective solution might be reached.
4)to design, construct and iteratively refine mock-ups and prototypes that constituted movement towards an effective solution.
5)Formal evaluation of a final prototye.
6)An informal investigation of the final prototype's potential applications.

Introduction & Need Statement

The standard QWERTY keyboard has become a major limiting factor in individual human-computer interaction. The common keyboard constrains mobility, impedes manual multitasking, and threatens human health. The shortcomings of the standard keyboard interface are made obvious by several current trends:

Trend 1: Virtual Office, Mobile Workers & Personal Digital Assistants

The nature of work and work environments is changing. There is a growing demand for "knowledge-based work", work that requires access to information, but does not demand a set work environment. Advances in technology are making nomadic work possible, and a growing number of people are choosing to work on the fly. According to a recent article in *Communications International*, "75% of professional managers and knowledge-based workers [are] able to [work while] on the move "(22). A 1998 study by Lasalle Partners Inc. forecasts that "Telecommuting [will] more than double in the next three years, and employee participation in the virtual office concept is likely to increase by about 77%"(23).

The ultra-mobile Personal Digital Assistant (PDA) computers fueling the shift toward mobile work patterns represent one of the most dynamic, fastest growing areas in personal computing. According to DataQuest Inc, "world-wide handheld computer shipments are forecast to exceed 5.7 million units this year [1999] a 47% increase over 1998"(24). By 2003, PDA sales are expected to reach 7.2 billion (US)(24).

A standard QWERTY keyboard can be reduced in size only so far before effective typing becomes impossible. As the number of PDAs too small to accommodate standard keyboards grows, the development of alternate methods of text entry becomes crucial.

Trend 2: The Repetitive Stress Injury Epidemic

Standard keyboards force a user to maintain a set posture while typing, and this requirement can stress the body in unhealthy ways. The long term effects of this stress--poor muscle tone, poor blood circulation, back pain and repetitive stress injuries (RSIs)--can be physically and psychologically devastating(12 & 27). OSHA predicts that this year, "nearly 3 million workers will file...claims for RSI and back injuries"(17). The number of reported RSI cases rose 42% between 1978 and 1990, according to the U.S. Bureau of Labor Statistics(27). Considerable financial cost accompanies the physical and psychological consequences of RSI occurance. According to OSHA, compensation "will cost employers...more than $20 billion(US) in direct costs and maybe another 100 billion in indirect costs"(17). The ergonomic shortcomings of the standard keyboard interface are partially responsible for the widespread RSI epidemic.

Trend 3: The Growing Importance of Multitasking

Modern living favors the ability to multitask. The linear, industrial, relax-and-read-the paper-on-Sunday-afternoons past has given way to a non-linear, post-industrial future where channel-surfing, eating breakfast and perusing the newspaper all coexist comfortably within the same moment. Multitasking is not just a preference; it is a prized skill in modern working environments. According to Bennie Thayer,

President of the National Association for the Self-Employed, "The ability to perform many functions at once is absolutely necessary"(26). Multitasking is especially valuable in scenarios where human and computer talents are combined: According to Microsoft, at any given time, a typical office user will run more than three programs...simultaneously(14). In an era of extreme multitasking--when "the inability to walk and chew gum [has become] cause for derision"--the fact that efficient use of a standard keyboard requires "all two" of a user's hands is a disadvantage (1, 167).

Trend 4: Intollerance Toward Invasive, Pervasive Technologies.

The rapid proliferation of digital devices without thorough knowledge or consideration of social/psychological factors has aggravated inclinations toward technophobia which must be addressed within any effort to design an improved text-input interface.

Digital devices demanding human interaction appear in a wider variety of contexts with greater frequency daily. Given that 1)the design of these tools often takes place without sufficient knowledge or consideration of context, and 2)there exists an undercurrent sociological anxiety related to computer control and human identity (see Appendix 2), new digital devices are often perceived to be annoying, disturbing and invasive. Miniature wireless headsets introduce the social discomfort of not knowing whether someone is in a local or remote conversation. PDA use in conferences sparks accusations of "video gaming" while on the job. Cell phones are now banned from many schools and restaurants, where people are weary of one-sided conversations and the intrusion of business in social and academic settings. As "smart" devices demanding human interaction infiltrate more

and more niches in individual and social life, the importance of developing interfaces that are socially and psychologically acceptable becomes crucial.

There exists a need for an effective text input device for computers that is highly portable, ergonomically correct, psychologically & socially acceptable and conducive to multitasking.

Beneficiaries

The primary beneficiaries of any solution effectively solving the problem previously stated include people who need to type while on the move, people concerned about preventing the onset of repetitive stress injuries, people engaged in activities where manual multitasking while typing is an advantage, and people who think & communicate best while not in a stationary environment sitting at static posture. This subset of the general population includes members of the following groups:

- On-site reporters
- The growing knowlede-based work force
- Delivery men and women
- Field researchers
- Secretaries
- Data entry personnel
- Programmers
- Businessmen
- Students
- New-media professionals

Secondary beneficiaries include loved ones of those at risk of developing RSIs, (the emotional and psychological strains brought on by intense carpel tunnel syndrome and other RSIs can be enormously taxing on a spouse or family), healthcare systems and taxpayers (who pay for recovery from RSIs). Other secondary beneficiaries include organizations depending upon a mobile work force: multinational corporations, social and economic development foundations, and the news reporting industry.

Great ideas are lost daily when not recorded. Children's author Roald Dahl (Charlie and the Chocolate Factory, James & The Giant Peach) recognized this, and attributes his imaginative works in part to his willingness to record ideas upon their conception, even if this meant stopping a car and jotting notes in the dust of the rear windshield when

paper and pencil were not to be found. Today, a great number of ideas (literary, musical and otherwise) pass through digital processing and conversion before actualization. The more convenient this translation process becomes, the greater an idea's chances of realization. The existence of a portable, highly usable typing solution would facilitate the preservation and communication of people's *inspiration* in addition to *information*.

Specifications

A sound solution to the need as stated fulfills the following specifications:

1) Portability

- **Weight**: <297g. (This value is based on the results of a weight survey that appears in Appendix 11).
- **Size**: <16cm maximal dimension, volume<14cm^3, Interface fits inside a small handbag or large coat pocket.
- **Durability**: Water resistant--capable of surviving one minute of operation in a shower. Shock resistant--will survive a 1m fall onto a concrete floor. Temperature resistant--operates within a range of 0-37^{0}C.
- **Range of Operation Between Interface and Computer**: Minimum range >2m, without intervening obstructions
- **Typing Session Duration**: The interface will support typing sessions > 6 hrs in length without the user having to stop to consider interface power requirements. (Six hours is longer than most people spend typing over the course of a day, according to survey results presented in the "Preliminary Research" section of this paper).

2) Health & Safety

- **Ergonomic Correctness**: The Interface allows the user's wrists and upper body to maintain neutral position. The interface does not demand static posture or restrain body motion. Actuation of characters does not require excessive motion or force. (See Appendix 3 for validation and explanation of these criteria).
- **Physical Safety**: The interface contains no sharp edges. Normal operation of interface does not

introduce toxins into the user's body or subject the user to radiation levels exceeding the limits outlined by the FCC.

3) Acceptability

- Wins head-to-head against any other design in the same survey group of potential users as the "most acceptable" solution.

4) Effectiveness

- **Learnability**: The interface's system of text entry can be learned by potential users--at a basic level--in 5 hours.
- **Speed**: After 10 hours of use, typing at 15 words per minute (including error-correction time) must be possible for the potential user. After 20 additional hours use, typing at an average speed of 30 words per minute (including error correction time) must be obtainable by a potential user. [NOTE: These figures are based on comparable learnability and speed figures given by the manufacturers of competing text entry systems such as Infogrip (The BAT), Handykey (The Twiddler), and DDH Software (Palm Pilot "WPM" system)].

5) Multitask-ability

- The interface will facilitate the user's ability to independently utilize voice, hands and body, while typing, to a greater degree than competing text entry systems.

6) Feasability

- It must be possible to develop and test at least 2 prototypes within real world constraints.

7) Financial Costs

- **Development Costs**: Development costs will not exceed $600US, presuming existing Thayer School resources.
- **Estimated Mass Production Cost**: The estimated cost of a product based on the final prototype must not exceed $100US--a price competitive with the prices of special feature text input devices currently on the market, such as the Twiddler and BAT(both $199US, from Infogrip and Handykey respectively). Product cost estimates will be made by acknowledged experts based on a hypothetical production run of 100,000 units.

Additional Considerations

Although the quantified specifications will make possible measurement of the project's progress, these specifications will not be the only factors weighed within the project's decision making process. Several additional considerations that resist quantification will influence design decisions.

Simplicity:

Design will adhere to the Occam's Razor principle:the simplest solution tends to be the best.

Extendability:

Where possible, the design will invite extension in the way of increased functionality and a wider range of potential applications.

Psychological Considerations:

Given the psychological considerations discussed in the introduction and elaborated upon in Appendix 2, it is important that the nonverbal language of the interface's design communicate a sense of control to the user, and NOT a sense of being controlled. To this end, the interface must not physically envelop the user's appendages in the way that chording keyboard gloves and virtual reality gear do.

Background & State of the Art

A number of text-input interfaces exist which partially solve the need for a mobile, ergonomic typing solution. Presented below are brief descriptions of the functionality, strengths and weaknesses of some of the most promising text entry interfaces that are currently available and will be available in the near future.
[NOTE: Many of the input interfaces discussed are packaged within computing devices with output,input and processing capabilities. Only the input aspect of each device will be discussed, as the scope of this project does not include output or processing.]

Current State of the Art

A Miniature Keyboard

The Series5MX, Psion

This well-engineered mini-keyboard partially addresses portability and ergonomic issues, but does not facilitate manual multitasking while typing.

Advantages:

- Extremely well designed miniature keyboard

Disadvantages:

- Requires a stationary surface
- Fingers obscure view of keys

A Miniature Thumb Keyboard

The "Blackberry" Interactive Pager, Research In Motion

The Blackberry thumb keyboard is conceptually similar to the Series5 mini-keyboard, however its keys are actuated by thumb presses. This leaves fingers free to support the unit, thus typing without a stationary work surface is possible.

Advantages:

- Portable
- Compact
- Does not require a stationary work surface

Disadvantages:

- Slow typing speed
- Thumbs obscure keyboard view
- Operation requires two hands
- Requires all but advanced users to look at the keyboard

Handwriting Recognition

The Palm Pilot "Grafiti" Interface,3Com

The Grafiti Interface is an example of "half way" handwriting recognition: It recognizes highly stylized characters written with a stylus on a touch-screen.

Advantages:

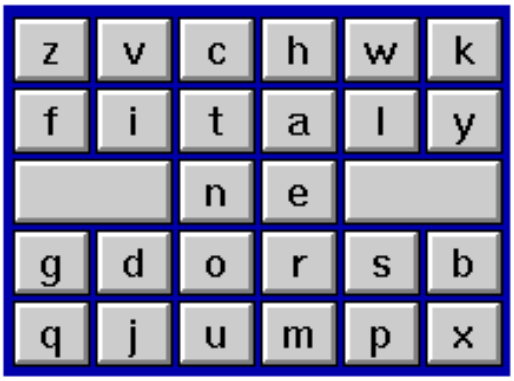
- Very easy to learn
- Similar to handwriting
- Portable

Disadvantages:

- Slow text entry speed
- Requires two hands
- Skill and concentration required

Stylus "Soft" Keyboard

The "Fitaly" Interface for the Palm Pilot, Textware Solutions



The "Fitaly" text entry interface for the Palm Pilot is a touch-screen keyboard designed to be operated with a stylus. It is optimized to minimize key to key finger travel, and places frequent letters near center-screen.

Advantages:

- Easy to learn
- Faster and more accurate than handwriting recognition
- Portable
- Minimizes hand motion

Disadvantages:

- Requires user to look at keyboard
- Requires two hands
- Limited character set; does not allow for punctuation or navigation

Voice Recognition

Dragon Naturally Speaking, Dragon Systems

Voice recognition has become a feasable, powerful way to enter text in recent years. The Dragon Naturally Speaking system requires a training period during which adaptive software algorithms "learn" how to accurately interpret one's speech patterns as text.

Advantages:

- No hands required
- Allows manual multitasking

Disadvantages:

- Requires a quiet environment
- Can't multitask vocally while typing
- Maintaining privacy becomes an issue
- Not currently a portable solution
- Initial training required. Training can be expensive--for business units it typically costs on the order of $1,500(34)

Desktop Chording Keyboard The BAT, Infogrip

The BAT is a one-handed typing solution. Key combinations--or "chords"--actuate characters. The design makes possible manual multitasking with the off hand, but is not portable, and has significant ergonomic flaws.

Advantages:

- Requires only one hand
- Bare bones simple design capable of initiating any keyboard action

Disadvantages:

- Initial training required
- Not portable
- Requires a stationary surface
- Keys provide no tactile response before bottoming out

Hand-Held Chording Keyboard & Mouse

The Twiddler, HandyKey

The Twiddler facilitates manual multitasking, and greater mobility than a standard keyboard (it is a wire-tethered device). The Twidler's versatility comes at the expense of simplicity and sound ergonomic design. The physical keyboard interface is significantly more complicated than that of the BAT, and use of the Twiddler does not allow a neutral hand position.

Advantages:

- Does not constrain body posture or position
- Allows manual multitasking
- Great one-handed solution to keyboard & mouse needs

Disadvantages:

- Initial training required
- Limited portability
- Poor ergonomic design
- A complicated plethora of actuator switches

Near-Future State of the Art

In addition to the interface solutions currently on the market, there are a number of text-entry interfaces in development that address aspects of need for a portable ergonomic typing solution that facilitates multitasking. The interfaces below represent some of the best solutions that may be available in the near future.

Virtual Keyboard Receiver

Berkeley Sensor & Actuator Center

Ten battery powered MEM accelerometers glued on finger and thumb nails transmit position information to a desktop receiver, which is then decoded as characters. Some interface designers believe the best interface is no interface at all. This typing interface comes close to that supposed ideal.

Advantages:

- Does not constrain body posture or position at all
- No mechanical parts to wear out.
- Allows for manual multitasking

Disadvantages:

- Key presses are not accompanied by tactile feedback
- Is a wired device--not portable
- Fingernails grow
- Does not yet exist--still a hypothetical solution

One Handed Mobile Chording Keyboard

The Data Egg, E2 Solutions

The Data Egg's text entry interface is a one handed, wireless, chording keyboard which allows greater mobility and manual multitasking than most other interfaces.

Advantages:

- Portable
- Compact
- Wireless
- Allows manual multitasking with the off hand
- Does not constrain body motion

Disadvantages:

- Initial training required
- Ergonomics have yet to be optimized
- Hand fatigue after 1/2 hour use
- Is currently in development and not yet available on the

- Requires only one hand
- Extremely flexible and extendible

mass market

Summary

Each of the above solutions addresses one or more aspects of the problem, but no solution currently available offers a comprehensive solution to the compound problem of mobility, multitasking & ergonomics in text entry. The Data Egg is the best solution presented, however there is room for improvement regarding its ergonomic design.

Path to a Selected Alternative

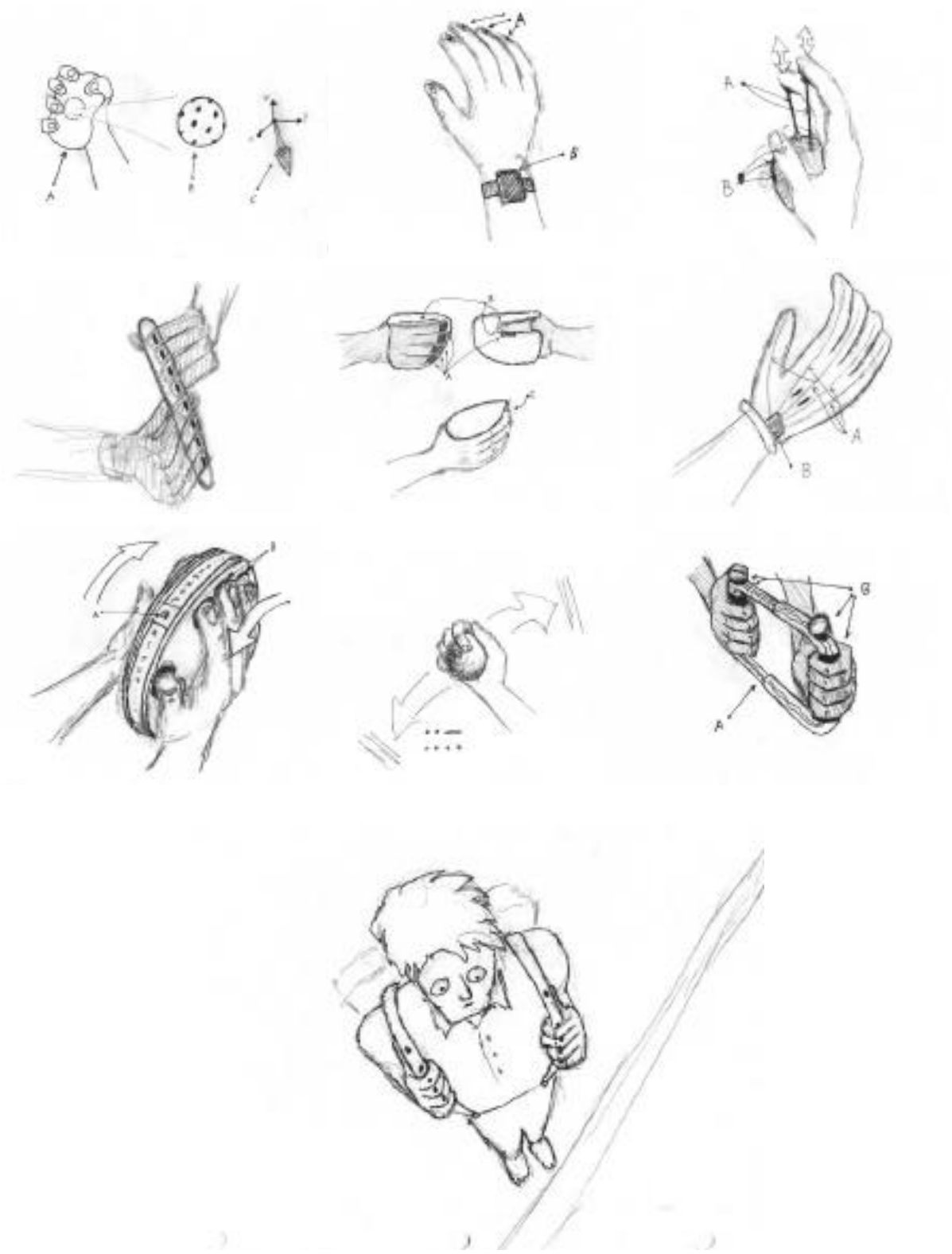
The process of choosing a potential solution to implement involved first brainstorming potential solutions, second evaluating these alternatives, and finally, choosing one potential solution for further development.

1) Brainstorming

> ***"Reality Bats Last"***
> ***--Alan Cooper,***
> ***The Inmates are Running the Asylum***

The path to a selected alternative did not chronologically begin with a brainstorm, however when the project ground to a halt and creativity ran dry, a brainstorm became necessary. The act of brainstorming--imagining various possibilities without regard to real world constraints--created a greater volume of options to consider. This greater volume improved the chances of choosing a great solution to focus upon.
Shown on the following pages are thumbnails of concept sketches illustrating some potential solutions. Full size sketches together with explanatory text appears in Appendix 4.

Concept Sketch "Thumbnails"



2) Evaluation of Alternatives

In order to facilitate the decision-making process, the project was parsed in two halves: "Human Side" and "Device Side". The human side of the interface consists of all a potential user touches, sees, feels, and interacts with, while the device side concerns the underlying electronic implementation.

Human Side Decision Path

Choosing a human side implementation involved making a number of decisions. For each decision, the alternatives were evaluated using a decision matrix. The structure of the decision matrices is as follows: Alternatives appear on the X-axis of an XY table, while design considerations appear on the Y-axis. Each alternative/consideration cell contains a number representing the alternative's "score" with respect to the associated design consideration. (Scores represent the guesses of one person). The totals of each alternative score row are summed--taking into consideration different numeric weights for different specifications--and the alternative with the highest total is deemed the "best" solution.

The human side decisions and their outcomes are shown below, while their corresponding matrices appear in Appendix 20. Selected alternatives appear in red, bold face type.

1: INTERFACE METHOD?

- Voice
- Eye movement
- Body movement
- Nerve Impulses
- **Hand/Finger Movement**

2: INTERFACE TOPOLOGY?

- Wearable (suspenders, gloves, etc.)
- No Physical Interface
- Handheld, two handed
- **Handheld, one-handed**

3: ACTUATION TYPE?

- Linear motion
- Linear force
- Discrete motion
- Discrete force
- **Discrete motion & force**

4: DISCRET CONTROL SCHEME?

- Serial Code (example: Morse code)
- Rotary Selection (like a label maker)
- One Actuator per Character
- Combinatory Actuation (Chording)
- **Rotary Selection with Combinatory Actuation**

[NOTE: Initially, the fifth option, Rotary Selection with Combinatory Actuation, was not considered. Through development of the first prototype--which relied solely on chording--this fifth option was "discovered" and subsequently used to create the final prototype.]

5: ONE-SIZE-FITS-ALL OR ADJUSTABLE?

- One Size Fits All
- **Adjustable**

6: INITIAL CHORD-TO-CHARACTER MAPPING?

- Adapt a pre-existing one-handed chord map
- **Develop a new mapping scheme**

[Note: This decision is a consequence of choice #4, thus it has no corresponding decision matrix].

Device Side Decision Path

Device side decisions were made using the same approach as human side decisions. Their outcomes are shown below, and their corresponding matrices appear in Appendix 20. Selected alternatives appear in red, bold face type.

1: DIGITAL DESIGN APPROACH?
- VLSI
- State Machine & Combinational Logic
- **Micro-Controller and High Level ICs**

2: TETHERED OR WIRELESS?
- Wired
- **Wireless**

3: WIRELESS MEDIUM?
- Infrared
- Ultrasound
- **Radio**

4: PROCESSING LOCATION?
- Handheld Unit
- Device Being Interfaced (via software)
- **Base Station Unit**

5: PROCESSOR ARCHITECTURE?
- USB-specific processor
- **8051**

6: FIRST PORT?
- MacADB • USB • AT • **RS-232**

7: SECOND PORT?
- MacADB • USB • **AT**

8: AT PORT METHOD?

• Tap into key matrix of existing PC keyboard
• Pipe RS-232 data through a software or hardware "wedge" to get PC-AT signals. (Wedges are commonly used to import data from bar-code readers).
• Learn the PC-AT protocol

3) Selected Alternative

The outcome of human side and device side decision making proceses was the following solution:

A one-handed, hand-held interface operated via finger motion and force. It uses chording and rotary selection in combination to actuate the transmission of characters from man to machine.

This wireless handheld interface will send information via radio signals to a base station 8051 architecture microprocessor circuit that maps finger chords to characters, then sends these characters to a "dumb terminal" window via RS-232 communication. Once the above solution has been successfully implemented, a PC-AT keyboard port will be added.

Design & Implementation Methodology

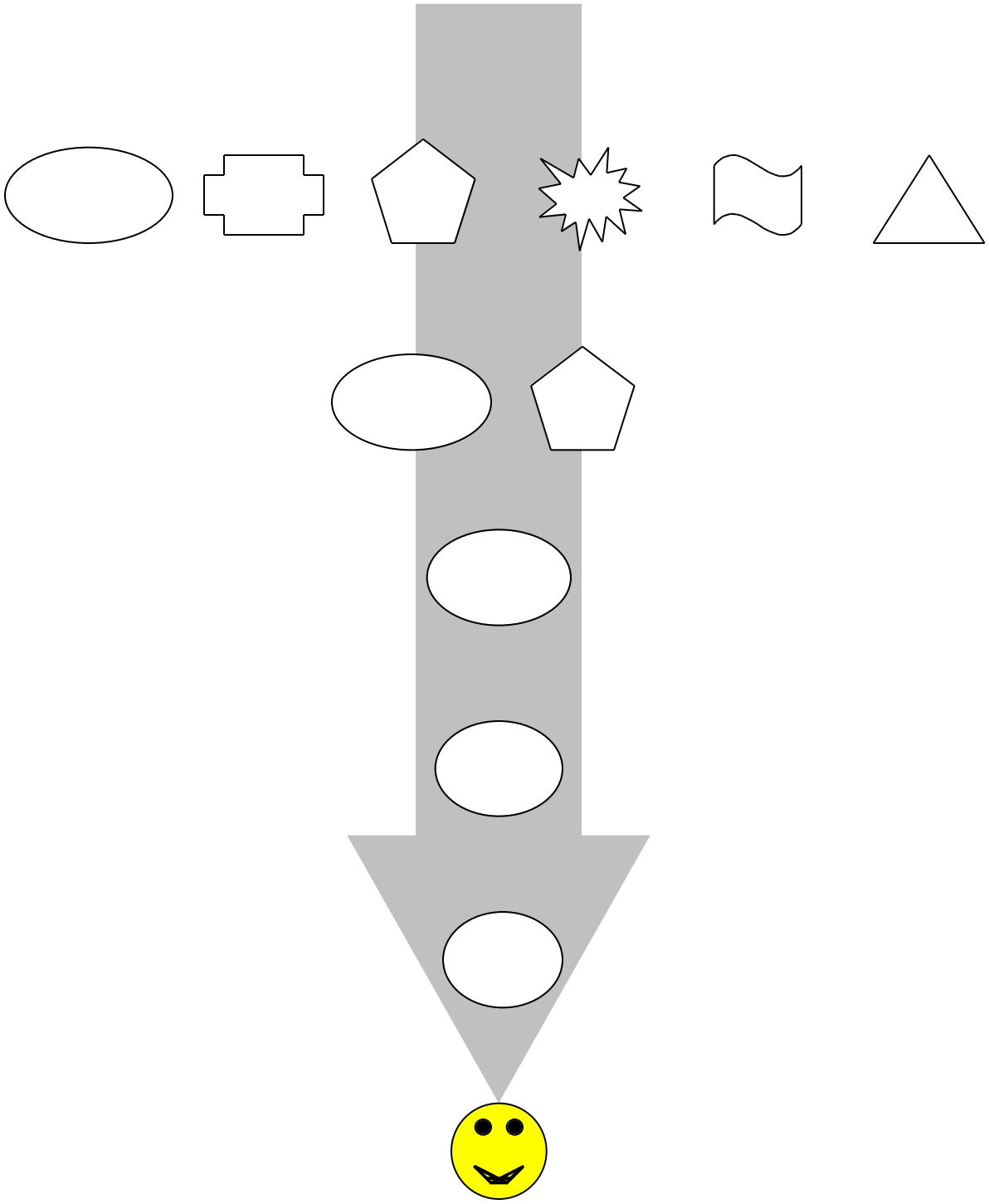
Human Side

Due to the number of variables and unknowns involved in ergonomics, interface design and man machine interaction, a heuristic, iterative model-based approach was chosen for the human side design process. This methodology pits numerous designs with different traits against each other, and permits a solution to emerge through a process akin to natural selection. (A general diagram of this process appears on the following page).

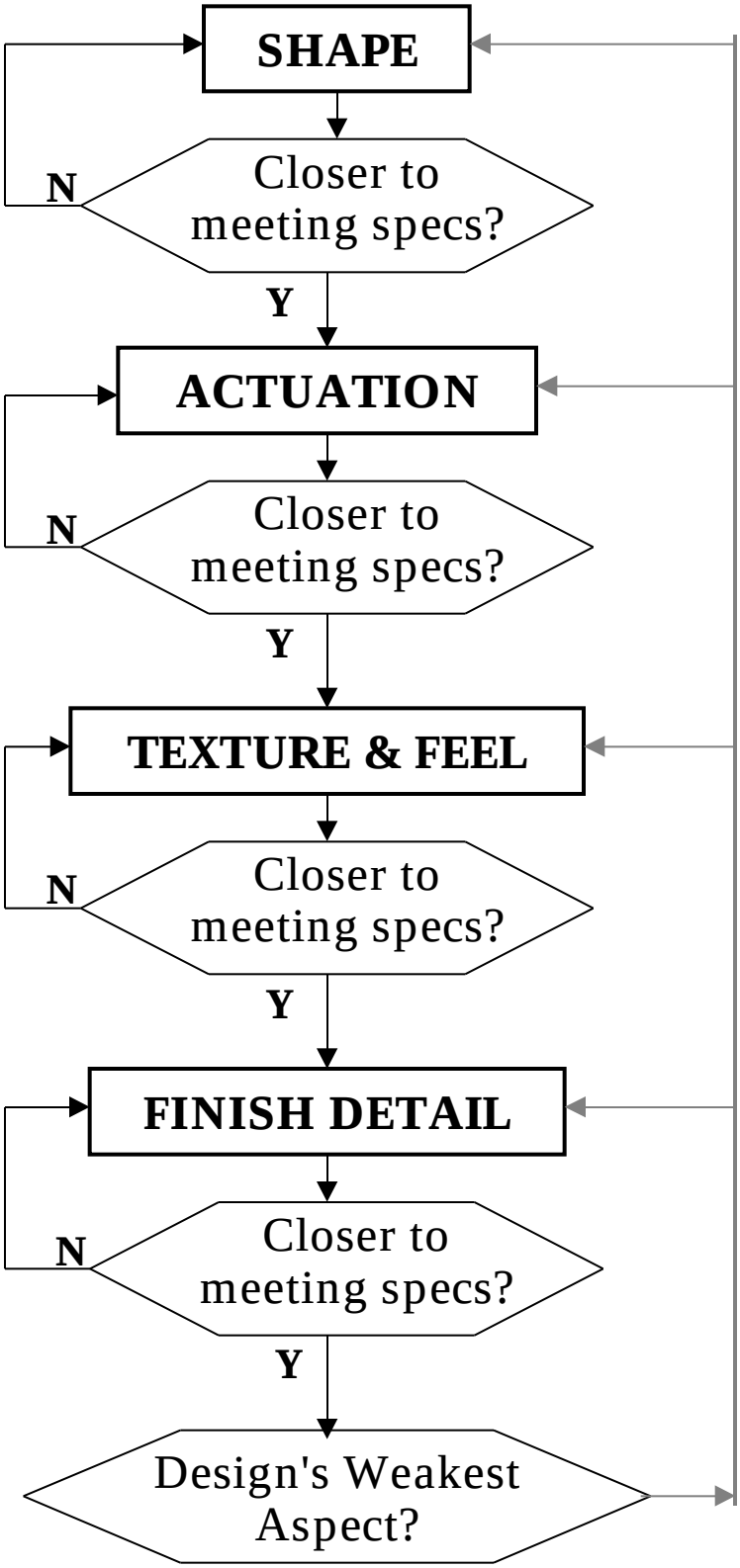
The specific iterative process for this project appears following the general diagram. This process involves the following steps: 1)developing a shape, 2)choosing a method of actuation and actuators, 3)determining the device's texture and feel, and 4)finishing touches. Between each step, the question: "Did the last step move the device closer to meeting specs?" will be informally answered. If the answer is no, the last step will be repeated until the answer is affirmative. During early iterations emphasis will be placed on shape. Later iterations will emphasize texture, feel and finishing detail. Later iterations will be formally evaluated according to the tests presented in the "Evaluation Methodology" section, while earlier iterations will be evaluated informally. The choice to informally evaluate earlier prototypes was made in order to a)make more time for prototype development, b)prevent test & survey subjects from becoming annoyed early on during the iterative development process.

Early iterations of the human side interface will be created through the use of foam mock-ups, epoxy molds and the process of "thermaforming": creating hollow plastic shells of mold forms through the application of heat and vacuum pressure. (See Appendix 7 for an elaboration on this process). If time permits, later iterations will be "reverse

Iterative Design Approach



Human Side Design Methodology



engineered": converted to mathematical models known as point clouds that can easily be scaled, manipulated and regenerated using CAD software and a rapid prototyper (3D printer).

Whenever possible, off-the-shelf components will be used.

Device Side

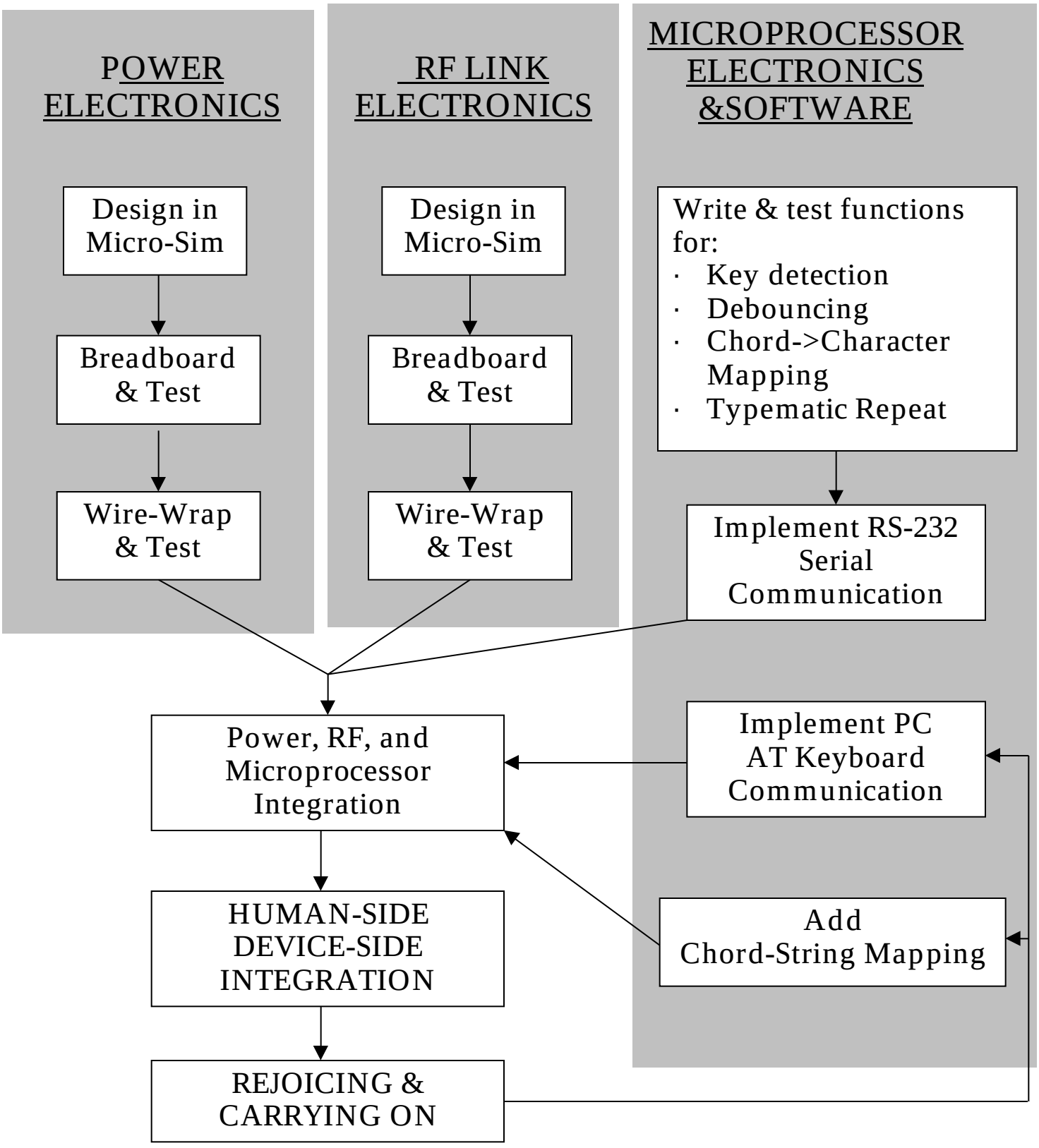
The device side design methodology consists of three convergent parallel efforts: 1) Power electronics, 2) RF link electronics, and 3) Microprocessor Electronics & Software. A diagram depicting these three efforts, their convergence and their integration with the human side interface appears on the following page. Modules will be debugged separately, and evaluated in conjunction with each iteration of the human side of the interface.

The underlying electronic implementation of the hand unit consists of a power supply, seven actuators, a parallel-to-serial encoder and a digital radio transmitter. Essentially, this circuit block is a set of switches, at a distance. A block diagram of this arrangement follows the "Device Side Design Methodology" diagram, and its derivative circuit schematics appear in Appendices 9 and 14. (These appendices correspond to prototypes 1 and 2 respectively).

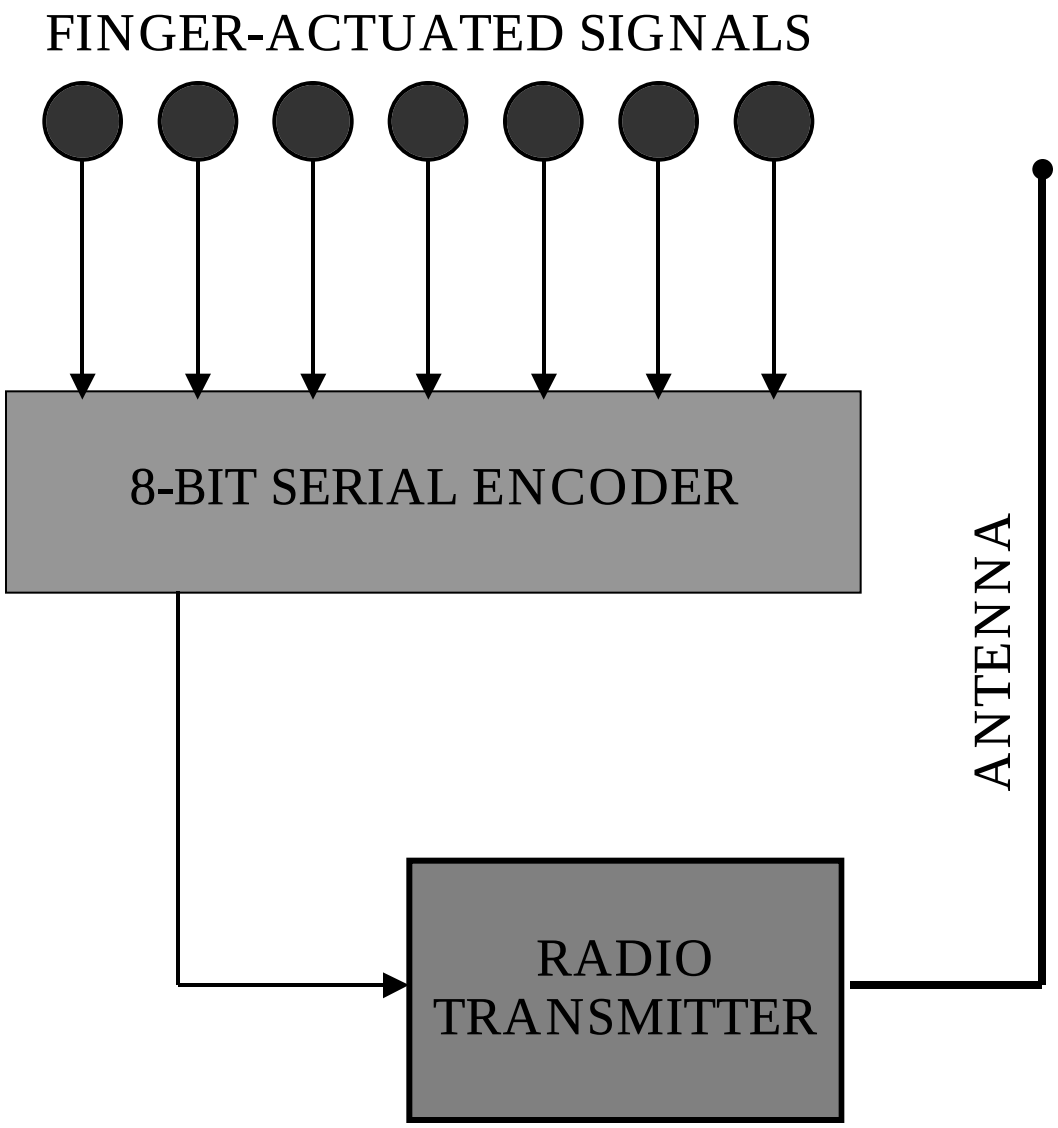
The electronic implementation of the base station is also relatively straightforward. The primary parts include a digital radio receiver, a serial->parallel decoder, a microprocessor and an RS-232 Buffer/Inverter. A block diagram of the base station's electronic implementation appears following the hand unit block diagram, and its derivative circuit schematics appear in Appendices 9 and 14.

During early prototypes, a "test board" for a class on microprocessors (ENGS62, Dartmouth College) will be used, since it already incorporates an 8051

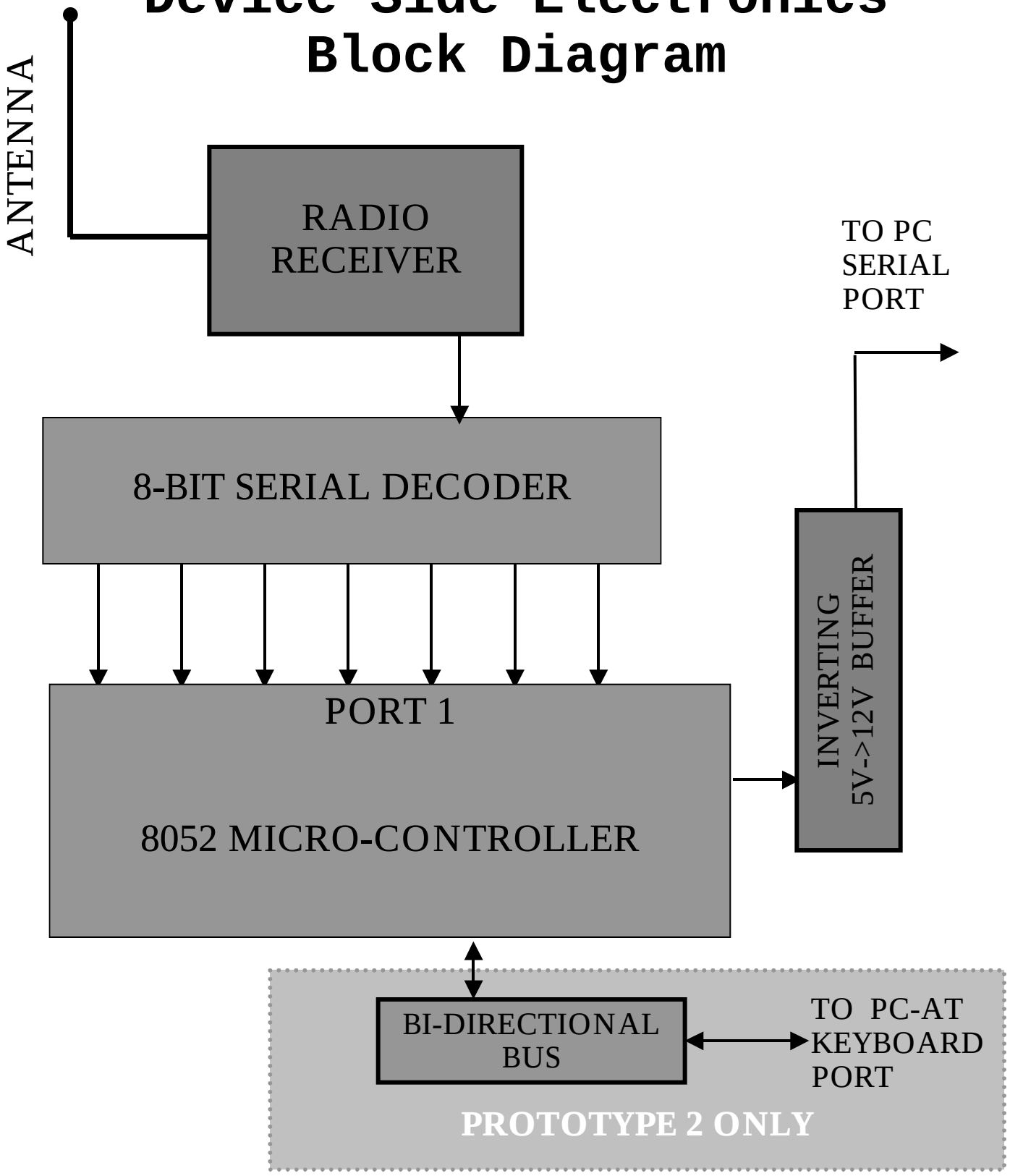
Device Side Design Methodology: Electronics, RF & Software



Human Side Electronics
Bock Diagram



Device Side Electronics Block Diagram



architecture microprocessor, serial communications hardware, and an operating system (MDP) for downloading firmware from a PC conveniently in one package.

Evaluation Methodology

The final prototype's quality will be evaluated according to the following set of tests, where ❑ signifies a binary test and ∠ stands for a linear test. Test outcomes fall between 0(low) and 5(high).

SPECIFICATION	EVALUATION TESTS	
PORTABILITY		
· Weight	❑ < 150g (weighed)	+ 1
· Size	❑ <16cm max. dimension (measured)	+.3
	❑ <14cm^3 (measured)	+.3
	❑ Fits in handbag & large coat pocket	+.4
· Durability	❑ Survives 1 minute of operation in a shower.	+.3
	❑ Survives a 1m drop onto a hard concrete floor	+.4
	❑ Survives operation at 0^{0}C and at 37^{0}C	+.3
· Range	❑ Operates @ 2m distance from base station	+ 1
· Typing Time	❑ Sends text for six hours continuously on one battery charge.	+ 1
		(max) 5
HEALTH & SAFETY		
· Ergonomic Correctness	❑ Hand maintains a comfortable resting position w/o pain.	+.5
	❑ Neutral wrist position	+.5
	❑ Doesn't demand static posture	+.5
	❑ Doesn't demand excessive digit motion or force	+.5
		+.5
	❑ Does not restrict body motion	+.5
· Physical Safety	❑ No sharp edges	+.4
	❑ Toxification not a significant risk.	+.3
	❑ Meets FCC specs for radiation safety.	+.3
		(max) 5

Evaluation Methodology (Continued)

SPECIFICATION	EVALUATION
ACCEPTABILITY	❑Wins head to head against other designs in the same survey group of potential users as "most acceptable". +5 (max) 5
EFFECTIVENESS · Learnability · Speed	❑Possible to learn typing system at a basic level in < 5 hours. +3 ❑15 wpm average speed possible after 10 hrs practice. +1 ❑30 wpm average speed possible after 20 hrs practice. +1 (max) 5
MULTITASK-ABILITY	While typing, ❑Can multitask vocally +1.5 ❑Can multitask with one hand +1 ❑Can multitask with other hand +1 ❑Can multitask with body +1.5 (max) 5
FEASABILITY	❑Two prototypes developed within real world constraints. +5 (max) 5

Evaluation Methodology (Continued)

SPECIFICATION	EVALUATION
COST	❑ Development Cost < $600US +2 ❑ Estimated cost of a product based on final prototype doesn't exceed $100US, given a hypothetical production run of 1000 units. (Cost estimates given by a panel of experts). +3 (max) 5

Time permitting, the final prototype's scores for each specification will be entered into a decision matrix along with the scores of current state of the art alternatives. A comparison of totals will indicate the relative success of the final prototype as a solution to the problem of mobility, multitasking and ergonomics in text entry.

Preliminary Research

Preliminary research involved a survey on typing habits, numerous interviews, consultations and conversations, and reading selections.

Background Survey

First, an effort was initiated to obtain data on people's typing habits and their attitudes toward the machines they use to type. This was accomplished through a small-scale survey, sent to a number of men and women, mostly between the ages of twenty and thirty. Most were involved in academics in one way or another, though some were involved in healthcare, archaeology, engineering or marketing. All happened to be of upper & middle class backgrounds, and all had had extensive education and experience working with computers. The survey, along with a compilation of its results, appears in Appendix 5. The results are summarized below.

Twenty-two people responded to the survey; 21 were right handed, 1 reported not having a dominant hand. The average of the respondent's ages was 27. The majority (11) reported that their primary learning style was kinesthetic; this learning style was followed in popularity by visual (5) and auditory (1) learning styles.

All people surveyed spent some time typing daily, and daily typing durations were distributed in the following way: 14%:.5to1hr, 36%:1to2hrs, 18%:2to3hrs and 32%:>3hrs. Seventy-seven percent of the survey respondents reported spending more time typing now than five years ago. Only 14% reported not knowing how to touch type.

Reported average typing speeds varied between 65 and 120 words per minute (WPM), with an average value of 66 WPM. All respondents reported using a standard keyboard for typing, and on a scale from 1(low) to 5(high) the average of the comfort ratings that

respondents assigned to their keyboards was 3.45. Two of the 22 respondents suffered from keyboard-aggravated RSIs.

The things that people liked about their keyboards included familiarity and key characteristics: quietness, force feedback, and light operating pressure. People's complaints included keyboard size, asymmetric hand use, awkward wrist position, keys jamming, hitting two keys at once, the need for a stable flat work surface, key noise, and numeric keypad placement.

Respondents envisioned a number of where typing was desirable but impractical given standard keyboards. These scenarios included the following: "When there is limited space and I'm on the move". "When I want to take notes standing up or moving about". "When I only have one hand". "When I want to record spontaneous ideas". "So I can eat and answer e-mail at the same time." "When I need to type equations." "When I want to type while travelling or working out". Respondents were asked to rate how valuable typing while on the move would be for them on a scale from 1 to 5(high), and ratings ranged from 5 to 1, with an average value of 3.25.

When asked whether or not the notion of a light-weight electronic device attached to their bodies or sewn into their clothing would bother them, fifty four percent of the respondents expressed concerns about these possibilities. The concerns expressed were: interference with style and fashion, the inability to get away from technology, the size, weight, expense and fragility of the wearable devices, interference with comfort, invasion of personal space, and the fear of being branded a nerd or geek.

Interviews/Consultations

Numerous experts were consulted in order to develop a sound design approach. Questions about ergonomics and repetitive stress injuries were brought to Lisa Tiraboschi, a member of the Dartmouth Environmental Health and Safety Group. Professor Peter Robbie's expertise was sought concerning the development of a sound design methodology. Conversations with Professors John Collier, Ted Cooley, Stu Trembley, Francis Kennedy, and Solomon Diamond helped to sharpen the project's focus and better define specifications. Research engineer Doug Fraser contributed advice regarding how to balance and coordinate human and device side development efforts. Graduate student Heather Wakely was consulted regarding the possible value of using computer simulation as a tool within the project's decision making process. Informal conversations with interface designers at Lucent Technologies, Sensable Technologies, Cognetics, and E2 Solutions also helped in the development of the project's design process.

In addition to facilitating the development of a design approach, expert advice was useful in making decisions regarding implementation. Professors Ted Cooley and Clayton Okino were consulted regarding the question of how best to implement the digital design portion of the project. Software engineer Michael Fromberger's aid was enlisted in deciding whether to map chords to characters within the interface device (as firmware) or within the computer being interfaced (as software). A later consultation with Michael Fromberger ensured that the microprocessor firmware algorithms developed for key detection and chord-to-character mapping were not unnecessarily complex. Professor Charlie Sullivan's advice was sought regarding power requirement issues and actuator characteristics. Professor Sullivan's experience with keyboard aggravated carpal tunnel syndrome allowed him to articulate the physiologically desirable characteristics of digit-operated actuator switches.

Pete Fontaine of Thayer School machine shop infamy was consulted throughout the development of the interface's physical shape. Specifically, Mr. Fontaine lent advice concerning modeling materials and molding techniques. Assistance regarding circuit noise resistance, troubleshooting and wireless communication alternatives was obtained from Professor David Stratton. The expertise of Frank Montegari at Glolab.com was sought regarding the implementation and troubleshooting of an encoded wireless radio linkage.

Reading Selections

Reading selections focused on typing machines, interface design, ergonomics, the nature of present-day work and the psychology of human computer interaction. For the purpose of learning about typing machines, the following works proved especially valuable: The History of the Typewriter, Century of the Typewriter, and Gramophone, Film, Typewriter (3 2 & 4). The most useful works on interface design included Interface Zen, Usability Engineering, Human Factors in Engineering and the Handbook of Human-Computer Interaction(29 30 31 & 32). Faster and numerous periodical articles lent insight concerning the nature of typing in present day work (1 etc.). The works most useful for the purpose of learning about the psychology of human computer interaction were: Gramophone, Film, Typewriter, The Inmates are Running the Asylum, How We Became PostHuman, Computer Anxiety and numerous periodical articles (4 10 7 11 etc.) (For a complete list of works consulted, see the "Resources" section).

Design, Development & Evaluation of Prototype 1

Human Side Development

Shape

On Professor Peter Robie's recommendation, the human side interface effort began with the creation of a large number of mock-ups. Before any formal evaluation, 12 prototype shapes were constructed from Eurathane foam. A friend contributed a 13th shape composed of plaster of Paris. This "family" of shapes appears in the photo below:

First Iteration of Potential Shapes

Once this first round of shapes was created, a survey was conducted to determine the strengths and

weaknesses of each shape as the shape of a potential typing solution, and to select two or three shapes to focus on for subsequent iterations. The survey and a compilation of its results appears in Appendix 6. A summary of the results appears in the table below.

[Note: The criteria presented in the table differ from the design consideration described in the "Specifications" and "Additional Considerations" sections because the survey was one of the tools used to develop these considerations.]

Shape Survey Result Summary

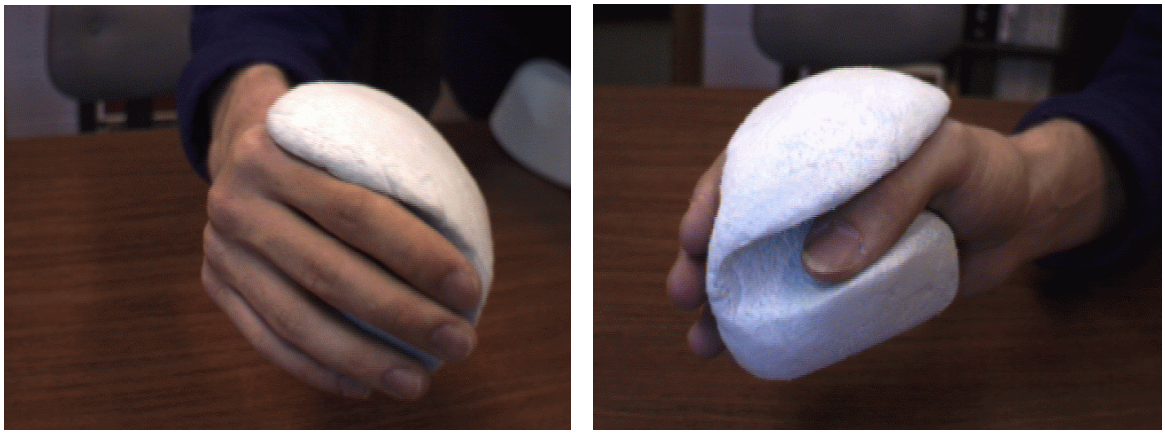
Shape-Secific: · **Most Comfortable** Scores on 1-5(hi) scale were averaged.	**Shape2** (4.2), followed by **Shape3** (4.0)
· **Easiest To Pick Up** Scores on 1-5(hi) scale were averaged.	**Shape9** (3.2), followed by **Shapes11&4** (2.9)
· **Easiest to Put Down** Scores on 1-5(hi) scale were averaged.	**Shape2** (2.3), followed by **Shape12** (2.25)
· **Easiest to Envision Typing With** % of respondents who could envision typing with shape.	**Shape2** (100%), followed by **Shapes3,11,&12** (75%)
· **Favorite** % of respondents who chose shape as favorite.	**Shape2**(50%) followed by **Shape3**(25%)
· **Most Comfortable** % of respondents who chose shape as most comfortable.	**Shape2**(50%) followed by **Shape3**(38%)
General: · **Tethered vs. Wireless** % of respondents who chose wireless	**Wireless**(100%)

Shape Survey Result Summary (Continued)

General (Continued):	
· **One handed vs. Two handed** % of respondents who chose one-handed	**One Handed** (75%)
· **Max. acceptable weight** Average based on holding a bag of quarters.	**150g**
· **Possible Device Names**	**DataWhistle, KeyGrip, KeyType, ErgoHand, DigeyBoard, HandKey, Palmer, TeleDactyl**

The shape survey confirmed intuition; in the estimation of their fashioner, Shapes 2 and 3 were clearly the best suited for typing, and this judgement was reflected in the survey results. Given the extensive amount of time required to conduct the survey and the fact that its results confirmed intuition, the decision was made to base further design decisions on intuitive judgement during early prototype iterations. The time and effort necessary for formal survey evaluations would be saved for final design iterations.

The shapes chosen for further development were shapes 2 and 3, shown on the following page:

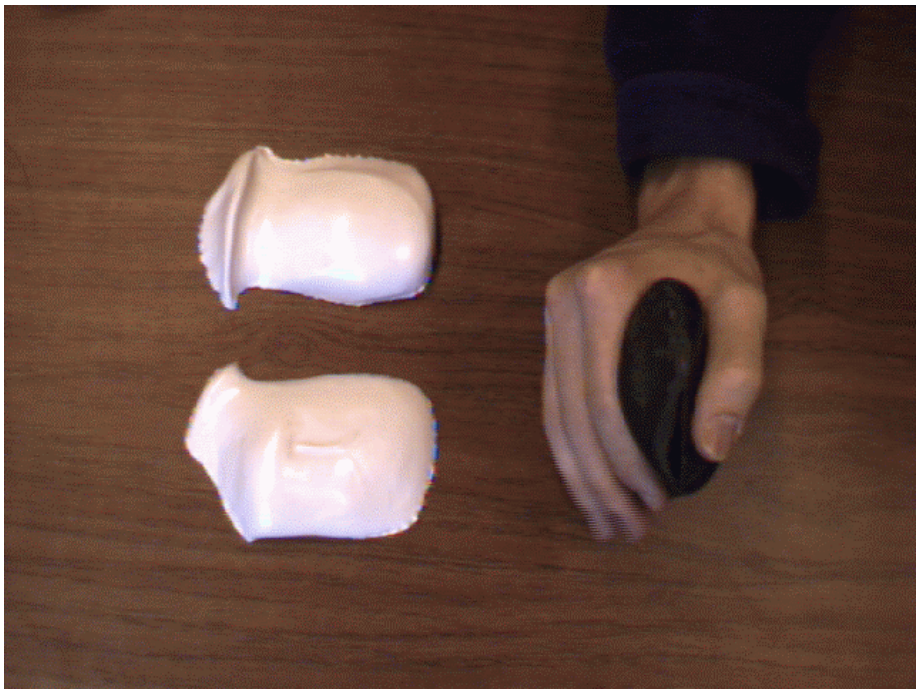


Shape 3

Shape 2

After two of the twelve shapes had been selected for further development, a plastic "shell" of one of the shapes, Shape3, was created using the Thayer machine shop Thermaformer. The Thermaformer "sucks" a heated plastic sheet over a mold in order to reproduce the shape of the mold. Creating this plastic shell involved cutting the chosen shape in half, laying both halves face up on a tray, and then positioning a flat piece of plastic above the shape halves. After the vacuum generated by the Thermaformer had sucked the plastic around the contours of the shape, it became evident that the reproduction was not ideal. The suction pressure had partially deformed the foam shape halves, and the thickness of the molded plastic was great enough to prevent accurate replication of sharp corners. As a

result of these problems, the first molded plastic shell was not an accurate reproduction of the original foam model, and did not conform naturally to the hand. A picture of the foam mold halves, together with the faulty first plastic shell, are shown in the photo below. The shell is black, while the mold halves are white.



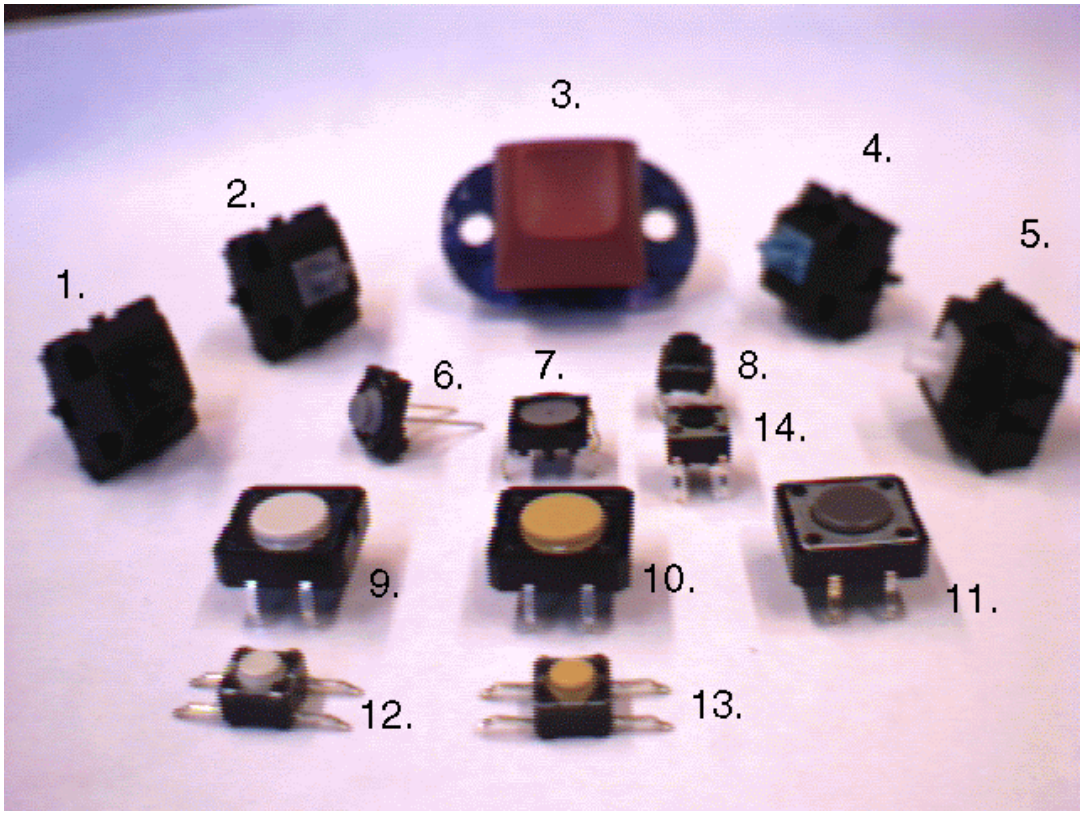
After the first effort to create a plastic shell, a new molding technology presented itself. Pete Fontaine suggested creating molds from a special two-part epoxy--"Apoxie®" produced by Aves®--instead of eurathane foam. Apoxie® can be shaped like industrial modeling clay during a four hour period, after which it hardens and can be cut, sanded and milled like hardened plastic. This material proved an ideal molding material for two reasons: Since it was harder than eurathane foam, it did not deform in the Thermaformer. Because the material could be formed by hand like modeling clay, the creation of shapes that felt "right" in the hand was facile.

Using the newfound molding material, two more shapes were created, based on the two best shapes from the first iteration (Shapes 2 & 3). One of the two new epoxy shapes was inferior to the two shapes that had inspired it. This shape was rejected. The other new shape felt more comfortable in the hand and allowed digits a greater freedom of motion than any previous shape. This shape was selected to be the shape of the first functional wireless prototype. The two epoxy shapes appear in the photos below. The photo on the left shows Shape2 from the set of first iteration shapes, a lump of modeling clay, and the rejected epoxy shape. The right-most photo shows Shape3 from the set of first iteration shapes together with the selected epoxy shape.

The selected epoxy shape from the right-most photo was halved and placed in the Thermaformer to create a new plastic shell. During this molding attempt, a thinner plastic was used so that the mold's corners and sharp edges would be more accurately reproduced. The result was a plastic shell of the desired shape, with fewer imperfections than the previous shell contained. The new shell held enough space for actuator switches, power supply and electronics. This shell appears in the following photo, together with its rejoined epoxy mold halves.

Actuators

Once an adequate shell for the interface had been created, development efforts turned toward actuator switches. The process of selecting actuator switches began in the same spirit as the process of selecting a shape: with many options, each with one or more distinguishing traits. Some actuators were procured as free samples from Cherry Corporation Electronics while others were bought. Shown in the photo below are a number of the actuators considered for the first functional prototype.



In the interest of quickly creating a working prototype before the end of the term, the simplest, smallest, most easily mounted switch, Switch6, was chosen for the first prototype.

Had there been more time, actuators would have been chosen based on criteria gathered from the background survey and conversations with professor Charlie Sullivan. These criteria were as follows:

Criteria for Actuator Selection

- **Tactile Feedback**: Actuators provide user with soft but discernable tactile feedback when contact is made.
- **Noise Level**: Actuators are quiet, if they provide any auditory feedback at all.
- **Actuation Force**: The force necessary to actuate is weak enough so that typing does not require great exertion, but strong enough to prevent inadvertent typing when picking up and putting down the device.
- **Consistency and Reliability**: Actuators have uniform characteristics, and perform consistently.

Device Side Development

Software

Development of the device's underlying implementation began with the development of microprocessor firmware which received as input seven finger & thumb actuator signals--one for each finger, three for the thumb--and sent characters as output to a dumb terminal via the RS-232 communication. The one-button-per-finger three-button-per-thumb configuration made possible the option of initially using an accepted, proven chord/character map developed by Dr. Daniel Gopher, an "internationally recognized expert in human factors and motor skills" for Infogrip(32). This decision temporarily narrowed the scope of the project without precluding a future effort at creating an optimized finger-chord/character map.

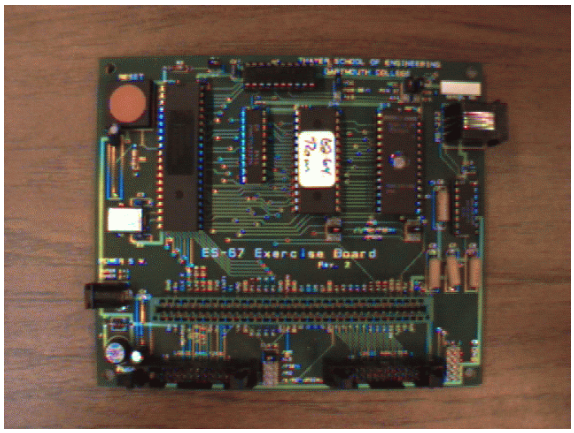
[Note: The chord map developed by Dr. Daniel Gopher was ultimately not used. There was not time to implement a full chord map for the first prototype; only a few chords were mapped to characters as proof of the concept.]

The microprocessor firmware was developed in C, using the Keil µ-Vision programming environment and compiler. The key functions developed were:

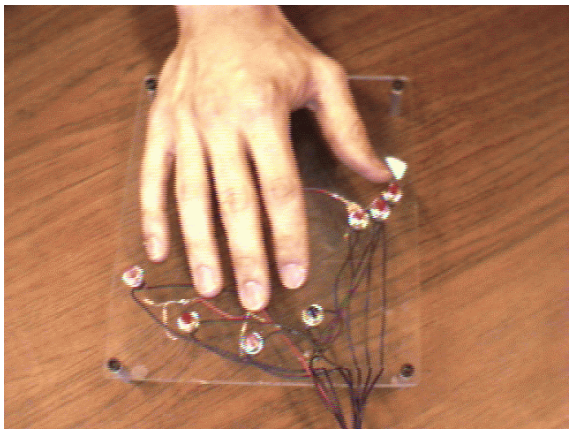
- **timer0_isr (void)** -- An timer driven interrupt service routine used to count 1 millisecond.
- **chord_is_changing ()** -- A routine which returns TRUE if the chord is changing.
- **settled ()** -- A routine that returns one when the current chord remains constant for a set ammount of time.
- **auto_repeat ()** -- A routine that allows "typmatic" action: the repeated typing of characters by keeping one chord pressed.
- **translate (byte chord)** -- A function that maps chords to characters.
- **send_char ()** -- A routine that sends a character from microprocessor to dumb terminal.

For complete code listings, see Appendix 8.

Once the functions had been written and various compilation annoyances worked out, the program was placed within the memory of an "ENGS62 Test Board": a circuit used for a course on microprocessors. This test board contained an 8051 arechitecture microprocessor and all the necessary hardware for serial communication, and thus served as a useful test platform. A picture of the ES62 test board appears in the left-most photo below. For debugging purposes, a "button board" was assembled in the machine shop, and connected to Port 1 of the test board's microprocessor. This wired keyboard appears in the right-most photo below.



ES 62 Test Board



Wired KeyBoard

Electronics

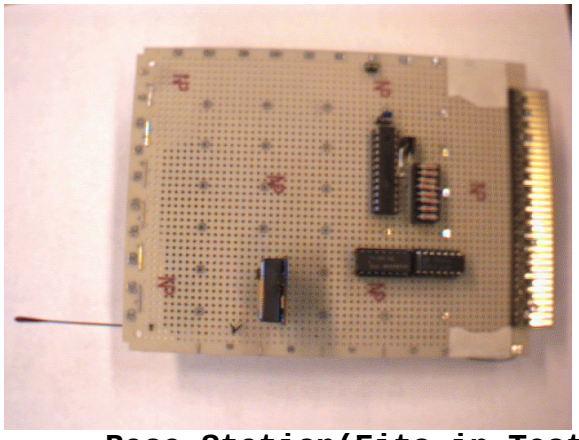
The principle challenge in the first prototype's electronic development was finding the necessary parts. After some searching, wireless transmitter and receiver modules were procured from glolab.com on the recommendation of graduate student Whitmore Kelley.

In order to separate human and device side development efforts, it was decided that the hand unit would contain switches-at-a-distance, and that the base station would be responsible for all signal processing. This decision meant that if at some point an extra chip were needed for processing, or

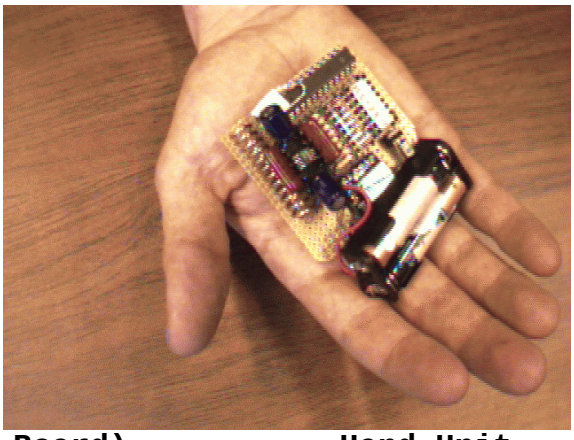
perhaps an extended chord/character map, the relatively tight size and weight restrictions imposed by the hand unit would pose no problems.

Two special purpose chips made the switches-at-a-distance implementation possible: an 8-bit parallel-to-serial encoders and its matching serial-to-parallel decoder (part numbers HT640 and HT648L respectively). These chips were discovered by Glolab, made by Holtek, and procured through Jameco. The encoder and decoder have several convenient features. They send information in bytes--one bit more than the 7 necessary for the first prototype. The chips send and receive information in an encoded fashion and do basic error checking--this prevents faulty transmissions or interference signals from being interpreted as valid chord combinations. In addition, the standby power requirement of these ICs is on the order of 1μA. This low power requirement, coupled with the comparable low power requirement of the wireless transmitter, made the need for an off/on switch on the transmitter effectively unnecessary--a convenient simplification for the user.

Block diagrams for the electronic design of the Prototype1 hand unit and base station appear in the "Design Methodology" section, while circuit schematics appear in Appendix 9. Photographs of the implemented circuits appear below.



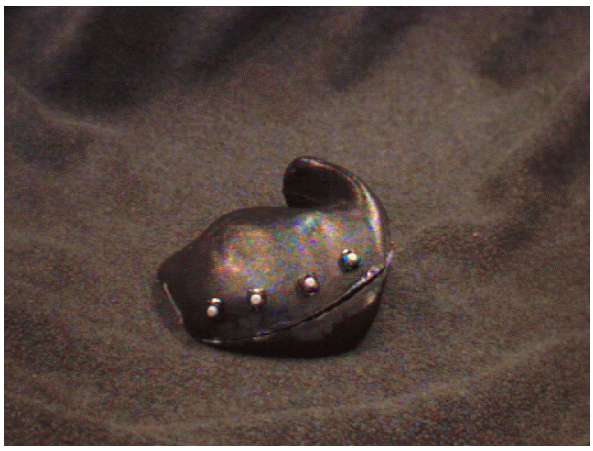
Base Station(Fits in Test Board)



Hand Unit

Integration

Once the software, human side interface and underlying implementation had been created, integration of the first functional prototype became possible. This involved connecting the shell's actuator switches to the hand unit electronics, plugging in the 3V AAA battery power supply, and temporarily sealing the shell halves with tape. The first fully functional hand unit appears in the four photos below.




Once the hand unit had been assembled, the base station circuitry was plugged into the ENGS62 test board, and the test board connected to power and a PC serial port. To the delight of this developer, the system worked!!! Different combinations of finger

presses resulted in the appearance of characters in a PC dumb terminal window, and remote typing could be done while stretching, walking about the room or having a coke.

Evaluation of Prototype1

The strengths of Prototype1 were that it demonstrated the feasibility of a wireless, one-handed text entry system. The hand unit was sturdy and its shape comfortable for various hand sizes. It was extremely light--so light that many people asked if the unit held electronics yet.

The weaknesses of the interface were numerous. Most of the people who informally tested the device found that operating thumb buttons while holding the device steady was awkward and challenging; it seemed that the user's thumb could be used to hold or to actuate, but not to do both simultaneously. Another design weakness was that the effectiveness of its buttons was critically dependent on hand size. Buttons had to be accessible to the user's fingertips, but there were great variations in user finger length. This factor made the positioning of buttons an issue of critical importance for future development efforts. Other less critical weaknesses included the device's aesthetics (it resembled an alien cow udder) and a data latching problem that caused the device to get "stuck" auto-repeating characters. (This latching problem is discussed in the following section).

Problems Encountered During the Development of Prototype 1

Expectedly, numerous problems were encountered. Unexpectedly, most of them were solved. The problems fall in three general categories: process problems, human side problems, and electrical/software problems.

General Process Problems

By far the most difficult problems encountered were problems of process. Given the developer's enthusiasm for the project, the wide range of potential applications and the number of tangents worth investigation, it was extremely hard to develop a clear, specific and concise understanding of what mattered. Numerous conversations and need statement revisions were required to clarify the design problem.

Quantifying specifications was another difficult hurdle. Since the project was in essence an interface design project, subjective judgement of potential users necessarily carried weight. In a sense, the user's subjective sense of the device's weight (for example) was more important than the device's actual weight. Because issues of perception and subjective judgement mattered, the project became a constant effort to balance engineering as an art on one hand, and engineering as a science on the other.

Another process problem encountered was poor survey design. The logistics of giving surveys and compiling survey data were not given proper consideration. As a result, time was wasted and nerves were frayed.

Human Side Problems

The problems encountered during human side development of Prototype1 concerned 3D visualization of shapes and troubles with the Thermaformer.

Creating numerous foam models helped make possible the visualization of complex organic 3D shapes. Unfortunately the foam models did not fair well in the Thermaformer. This problem was solved by making molds from epoxy resin instead of from foam.

In order to create plastic shells, models had to be halved. Halving the models using a band-saw removed a 2mm wide slice, and this alteration in shape prevented shell halves from lining up perfectly. This problem could be solved by adding 2mm of thickness to one half after splitting the mold in halves.

Electronic/Software Problems

Initially, the key detection code was written and compiled presupposing an 8051 microprocessor. The 8051 chip did not contain enough internal memory for both program and the 127-character array representing each possible finger combination. As a result, the compiler choked. The solution to this dilemma was to reconfigure the compiler for the chip actually being used: The Atnel 8052 microprocessor. This chip had double the internal memory of the 8051--enough to hold the 127-character array in addition to the program code.

One bug that provided great headaches was an unrecognized difference in decoder and encoder oscillator speeds. Oscillator speed for each IC was a function of one resistor value, and though two resistors of (supposedly) the same value were used, they were not within a close enough tolerance for decoder and encoder to cooperate. The solution was to obtain a pair of matched resistors.

The only serious bug in the circuit design of the first prototype resulted in the occasional tendency of the decoder to get "stuck" sending a chord to Port

1 of the microprocessor. This occurs when the transmission of the "no buttons pressed" chord fails after a successful transmission of a "some buttons pressed" chord. Since the decoder outputs were latched, the old chord values remain when the "no keys pressed" chord is desired. A solution to this problem was to place an octal D-Latch in between the decoder and the microprocessor's Port 1, and have the latch outputs reset when the decoder's (active high) "Valid Transmission" signal line goes low. The second prototype incorporated this solution.

Design Changes and Refinements

Several lessons were learned from the creation and informal testing of Prototype 1, lessons that affected subsequent design decisions. These lessons pertained to the role of the thumb with respect to the hand-held unit, variability in human hand dimensions and the weight of the hand-held unit.

Addition of a Thumb-Actuated Rotary Switch

One lesson learned from the first prototype was that the user's thumb had difficulty supporting the hand unit and operating its switches simultaneously. This observation had a substantial impact on the final prototype's design. Because of the awkwardness of the thumb's dual role, the decision was made to have the thumb exert control over a rotary "case selection" switch instead of pushbuttons. This decision had several benefits; it made possible improved grip support, simpler, more natural and less frequent thumb motion. The use of a thumb wheel also had cognitive advantages over the use of thumb-actuated push-buttons.

Better grip support: The use of a thumb wheel instead of push-buttons allowed the thumb to exert device-stabilizing pressure when necessary, with less chance of inadvertently changing the chord being pressed.

Simpler, more natural thumb motion: The rotary switch reduced and simplified thumb control motion. Instead of having to move the thumb laterally, then downwards--as was necessary for the operation of Prototype 1--control of the thumb wheel required contracting the thumb at one joint: a simpler, more natural motion.

Less frequent thumb motion: Use of a rotary wheel gave the thumb a unique role in the chord actuation scheme. Instead of functioning as a fifth chord-button actuating digit, the thumb could now be used to select from between modes (or "cases") that

together composed the full chord map. The task of case selection (such as selecting between upper and lower case letters) occurs much less frequently in most text entry situations than the task of character actuation. This translates into less frequent thumb motion than was necessary given the character actuation scheme of Prototype1. Reducing the frequency of thumb motion necessary for character actuation allowed the thumb to more effectively carry out its role as a grip support.

Cognitive Advantages: The use of a rotary thumb switch allowed the chord map to be broken into several cases, each case holding a subset of the set of characters. This grouping of characters in smaller subsets increased the usability and learnability of the design. Since each subset (case) of characters has fewer members than the full character set, fewer buttons are required to define a unique character than were required for the implementation of Prototype1's chord map. Decreasing the number of actuators and thus the number of chords makes learning the chord map less intimidating and more learnable. It is generally easier to learn a system piece by piece than all at once, and the addition of a case wheel facilitates a piece-wise learning process. The use of a thumb-operated rotary case selection wheel makes possible one additional cognitive advantage: feedback concerning the current case of the device. Through rotation, a different section of the case selection wheel is exposed to the user's thumb for each position of the rotary switch. By mounting tactile symbols corresponding to case along the case selection wheel's perimeter, it possible for the user to "feel" the current case through his or her thumb. This is an elegantly simple way to provide the user with feedback concerning the device's mode, a way that imposes no additional power requirements on the hand-held device.

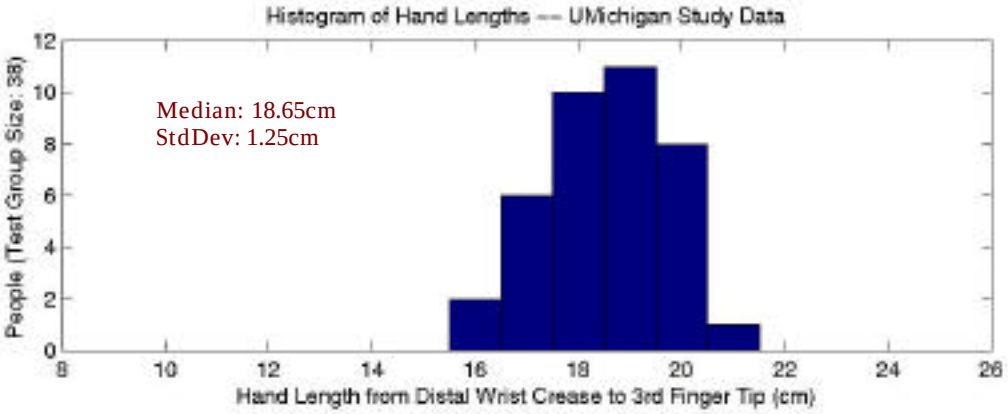
Hand Size and Adjustability

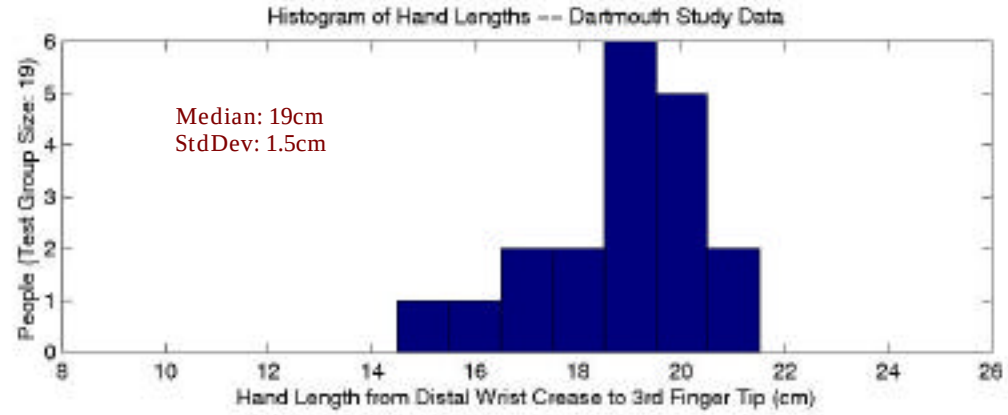
A second lesson provided by Prototype1 was that the usability of a chording system is critically dependent on hand dimensions. If buttons are not accessible to the user's finger tips, a chording system such as Prototye 1 becomes awkward--if not impossible--to use.

This problem brought up numerous questions related to hand size: What is an "average" sized hand? What does a "normal" distribution of hand sizes look like? Do the project's volunteer test subjects hands reflect this normal distribution?

To answer these questions, the internet was scoured for "amphometric" data: data on human body measurements. Data on hand size was found in a study conducted at the University of Michigan(38). This study measured hand length and breadth of numerous test subjects. The hand length data from this study was used to construct a histogram, and a second histogram was created using the hand lengths of this interface project's test subjects. These two histograms appear on the following page. Comparison of the two graphs reveals that the test subjects involved in this interface design project have hands of typical dimensions that vary over a range that is not out of the ordinary. (These conclusions presume that hand length is a representative hand size dimension, and that the University of Michigan measurements are accurate and representative of potential user's hand sizes.)

Hand Length Histograms





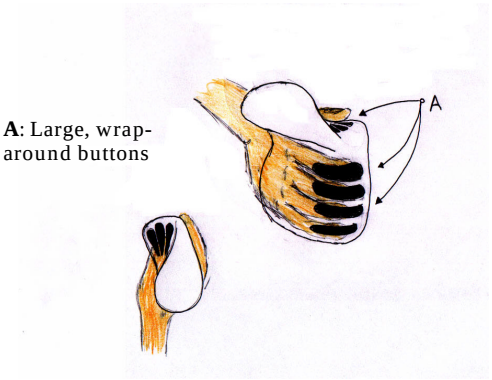
The problem of Prototype 1's critical dependence on hand size with respect to actuator placement created an ultimatum: Continue the development of a "one-size-fits-all" implementation, or create an adjustable solution. (The option of creating a person-specific solution was not considered, because it significantly reduced the quality and quantity of feedback that potential users could give.)

The advantages of a one-size-fits-all solution were simplicity and elegance. With few screws, wires and other implementation-specific artifacts visible to the user, a one-siz-fits-all solution would be more sleek and seamless. The disadvantages of continuing with a one-size-fits-all design approach were presumed to be the following: 1)dependence on specialized actuators with large, specially shaped touch surfaces, 2)a lack of knowledge concerning hand physiology, and 3)the risk that the final prototype would be a mediocre solution--something designed to fit the hand of a non-existent "average" user, but not quite right for any real user.

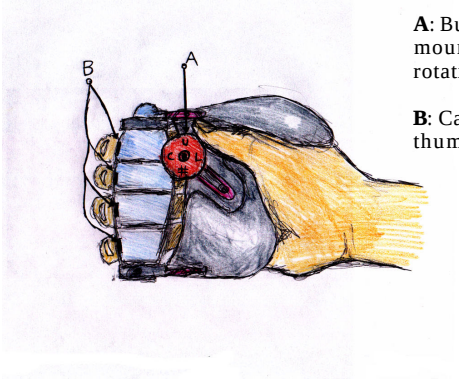
The advantage of continuing with an adjustable design would be configurability. Through configurability, hands of various sizes and dimensions would be accommodated. This would widen the range of actuators that could be considered, and meant that the design's physical dimensions didn't have to be "perfect"; problems could be compensated for through adjustment.

The disadvantages of the adjustable design were complexity and fragility. Creating an adjustable mechanism would require significantly greater amounts of thought and effort, and configuring the device could add complexity to the user's experience. Fragility was a risk because a greater number of parts would be required for an adjustable solution, and each additional part would bring new potentials for breakage.

Two concept sketches were created to illustrate what one-size-fits-all and adjustable solutions might look like, and these sketches appear below.



One-Size-Fits-All



Adjustable

The envisioned one-size-fits-all solution has wrap-around finger buttons, and thumb buttons that extend outward to accommodate digits of different lengths. This solution relies on switches with relatively large, specially shaped actuation surfaces.

The envisioned adjustable solution has buttons mounted on sections of a cylinder. Each section can be rotated to move the position of its button with respect to the palm. Through rotation, different length fingers are accommodated. The sketch of the adjustable solution also incorporates the case selection thumb wheel discussed earlier. This wheel is mounted on an adjustable arm.

After considering the pros and cons of the two alternatives, the decision was made to direct future development efforts toward an adjustable solution.

Device Weight

A relatively minor change in design to the design specifications concerned weight. The first prototype seemed incredibly light--its lightness made the hand unit feel insubstantial to several users. This unit was initially designed to meet a weight (mass)

specification of 150g--a figure based on data compiled from the Shape Survey (see Appendix 6). In this survey, a bag of quarters was placed in the hand of each participant, and participants were asked to add or remove quarters until the bag held--in their estimation--the "maximum allowable weight" for a one-handed typing device. 150g was the average weight from these test trials.

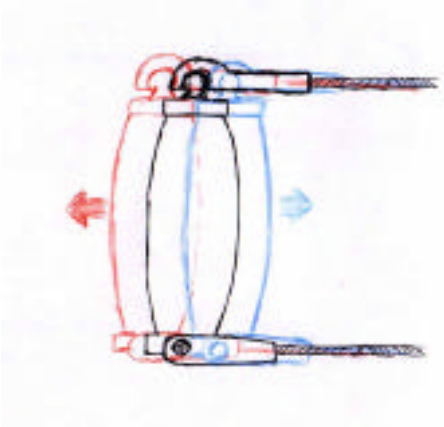
This test did not distribute weight across the user's hand, and as a result, the first prototype--which *did* distribute weight across the hand--seemed especially light. Because of the discrepancy between the 150g survey figure and the apparent lightness of the first prototype, the weight survey was conducted once more. During this second weight survey, the bag of quarters was placed in a thin plastic shell thermoformed from the same mold used to construct the final prototype. Doing so distributed weight over survey participants' hands, in a more realistic fashion. The average "maximum acceptable weight" figure jumped to 297g. To make a short story long, the apparent lightness of Prototype 1 brought to attention an oversight in survey implementation and influenced the weight specification used to judge the final prototype.

Design, Development & Evaluation of Prototype 2

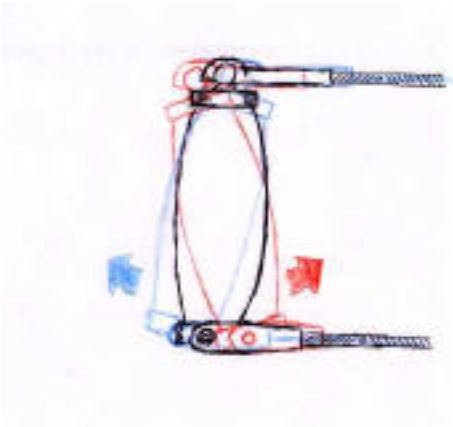
Human Side Development

After deciding upon a solution that used fingers for chording and thumb for case selection, it became necessary to develop an adjustable implementation. The following sketches illustrate the hand unit's manner of adjustability.

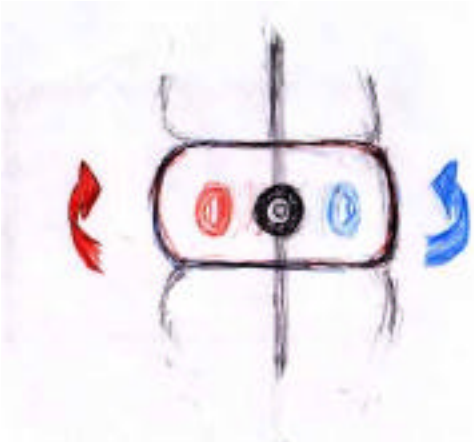
Final Prototype: Adjustability



Extension and Contraction of the Finger Button Column



Tilting of Finger Button Column



Rotation of Button Modules on Column

Extenson, Contraction & Rotation of Thumb-Wheel Arm

The first three sketches depict the adjustability of the finger button column's design, while the last sketch depicts the adjustability of the thumb wheel.

The finger button column is attached to the body of the hand unit via two turnbuckle tie rods. These tie rods are reverse-threaded on one end so that expansion and contraction can be achieved simply by twisting the tie rod--no disassembly or disconnection is required. The top tie rod is attached to the button column assembly via a ball joint, while the bottom tie rod connects to this assembly via a "U" shaped joint with one degree of freedom. This portion of the adjustable design is somewhat similar to the front suspension system of a formula car.

The thumb wheel is attached to the body of the hand unit via an arm. A milled slot runs the length of this arm, and by loosening the screw passing through this slot, the arm can be extended, contracted or rotated. Tightening the screw sets the position of the wheel in relation to the body of the hand unit.

The button modules on the button column are sandwiched between two plates that prevent them from rotating. By loosening a screw at the base of the button column assembly, the pressure between the two plates is reduced, and button sections can be adjusted to the desired angles. Re-tightening the screw sets the angle of the button units with respect to the hand unit's body.

On the following two pages are photos depicting the development of the final prototype's human side--from an initial foam mock-up, to the completed hand unit. The processes used were essentially the same as those used to make Prototype 1: Create quick-and-dirty prototypes to test concepts, mold structural parts using the Thermaformer, and iterate whenever possible. Care was taken to design and build the device's sub-assemblies in modular fashion to facilitate disassembly and repair. Due to the organic, hand-held nature of the shapes being constructed, CAD/CAM techniques were not employed.

Final Prototype: Human Side Development

Adjustable Mock-Up

Molds: Prototypes 1&Final

Switch Candidates

Finished Switch Modules

Button Barrel Evolution

Case Wheel & Candidates

Final Prototype:
Human Side Development (Continued)

Shape Evolution

Final Prototype

Final Prototype (R)

Final Prototype (L)

Open (Outer View)

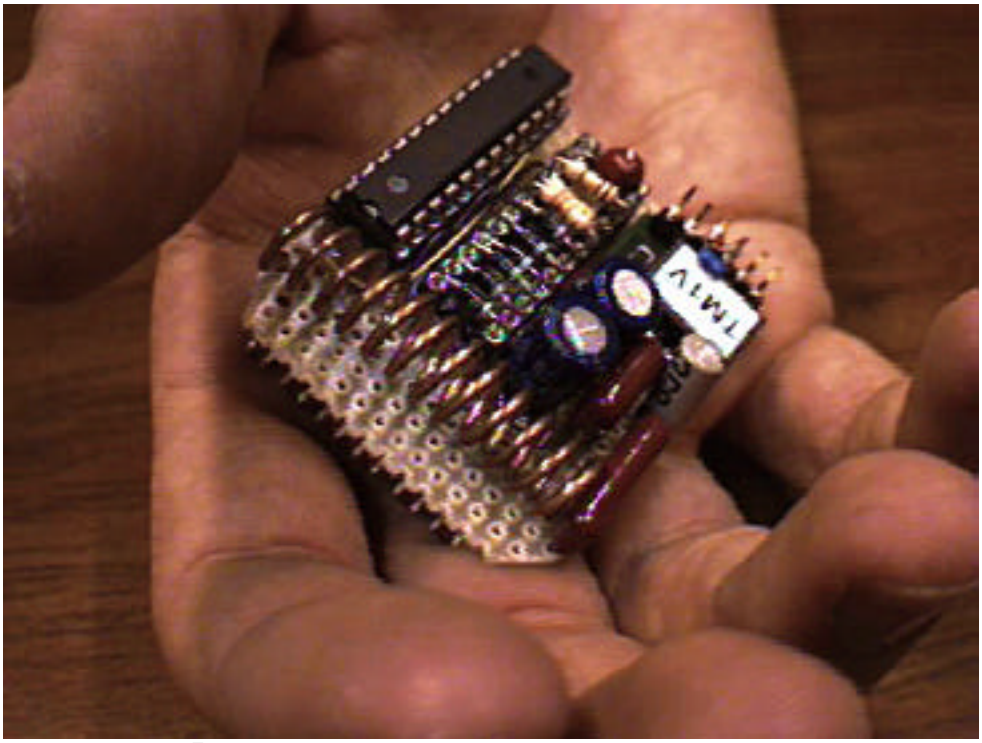
Open (Inner View)

Device Side Development

Hardware

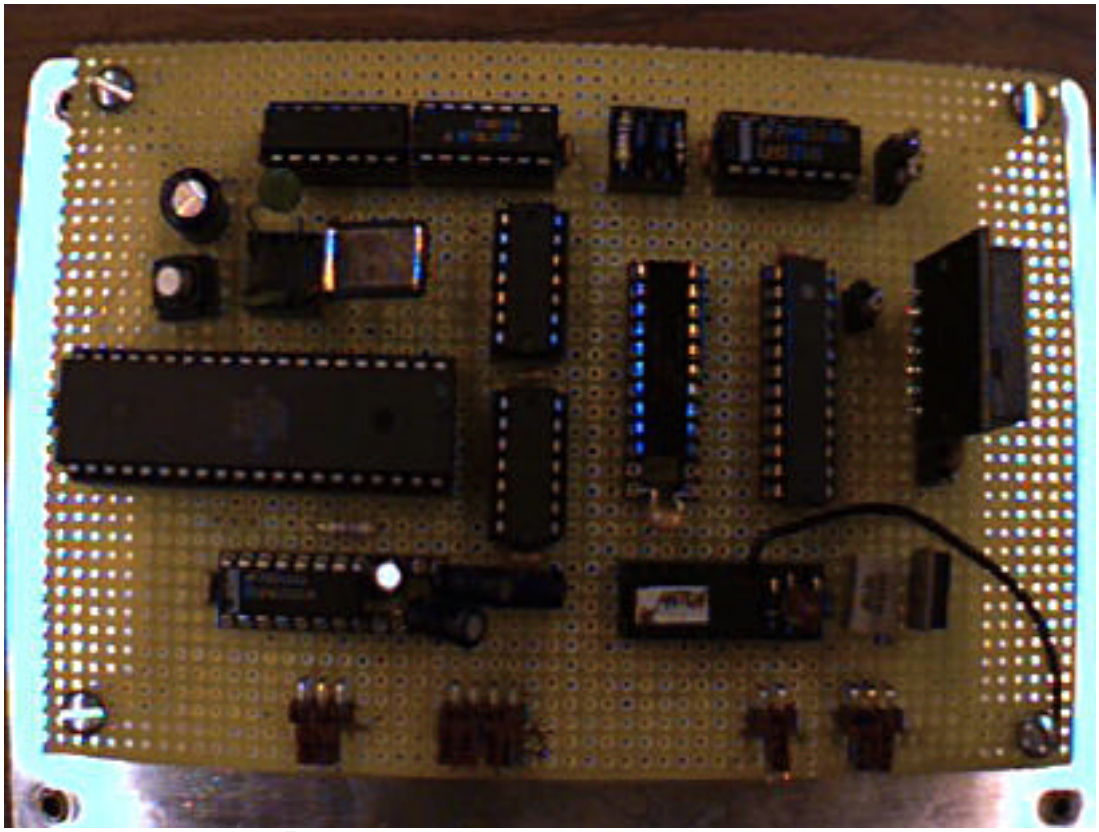
Hardware development for the final prototype involved refining the hand unit circuit and making a stand-alone base station circuit with RS-232 and PC-AT ports.

The hand unit was improved through simplification and miniaturization. It was discovered that the hand unit's encoder had *internal* pull-down resistors--a fact that had escaped attention during the development of Prototype 1. As a result of this discovery, fewer resistors were included in the final hand unit circuit. Through the use of fewer resistors and effective space management, the surface area of the hand unit was reduced by approximately 30%. A photo of the final hand unit circuit appears below. For a block diagram of this circuit, see the "Design and Implementation Methodology" of this paper. For a detailed circuit schematic, see Appendix 14.



Final Prototype's Hand Unit Circuit

Development of the final prototype's base station circuitry was more complex. A design bug had to be corrected and numerous functional blocks had to be added to the board--including a microprocessor, communications hardware and battery charging circuitry for charging the hand unit. A photo of the final base station circuit appears below. Its corresponding block diagram appear in the "Design and Implementation Methodology" of this paper. For a detailed circuit schematic, see Appendix 14.



The Final Prototype Base Station Circuit

The base station's serial->parallel decoder IC latched the serial data it received and as a result, finger-chord bytes were occasionally retained after their corresponding finger buttons had been released. This meant that the base station circuit would sometimes become "stuck" auto-repeating the character corresponding to the retained chord indefinitely. To solve this problem, an 8-bit latch with an asynchronous "clear" input was placed between the decoder and the microprocessor input port.

When the decoder's "Valid Transmission" (VT) output signal went high, chord data entered the latch. When the VT signal went low, the latch was cleared. The addition of this latch effectively solved the auto-repeat problem. One tricky aspect of implementing this solution was that the decoder's VT output controlled both "clock" and "clear" latch inputs, and a delay was required on the "clock" signal line in order to prevent simultaneous clearing and latching. This delay was created through the use of a buffer, a low-pass filter and a comparator.

Creating a stand-alone base station circuit board (independent of the ES62 test board that was necessary for Prototype1) involved adding a microprocessor and communications hardware. After a false start (buying a microprocessor with half the RAM, one too few internal timers and half the code memory necessary) the Atnel AT89C55 microprocessor was chosen. This 8051-architecture chip had all the necessary features, and twice the re-programmable code memory necessary. To implement the serial communications port, a 12V/5V inverting buffer (Maxim's MAX-232) was used. To implement the hardware of the PC-AT keyboard port(a bi-directional clock-and-data-line bus), a pair of buffers (really 4 inverters, two with open collector outputs) was used.

The base station circuit for the final prototype incorporated battery charging circuitry for the hand-held unit. Initially, fast-charge circuitry was desired, a circuit that would monitor battery charge and temperature in order to ensure fast peak charging as well as long battery life. With these goals in mind, a sophisticated charge-monitoring IC and temperature-sensing thermistors were acquired. Unfortunately, the fast-charge circuitry proved too difficult to implement within the project's time frame, and a simpler battery charging circuit was employed, a circuit developed and posted on the web by Ralph Tenny of the Dallas Personal Robotics Group(37). This charging circuit used a feedback-controlled voltage regulator to supply a battery with constant charging current. For the 300mAh hand unit's nickel cadmium battery, the charge current was 30mA, and the charge time

was 14 hours. The circuit required a 12V power supply, but was adapted to work (less effectively) with the 5V supply available on the base station circuit board.

Another addition to the base station circuit board was a ground plane, oriented perpendicular to the antenna. This was done to improve the range of RF signal transmission.

Firmware

The firmware used to run the final prototype's base station was essentially the same as the firmware used to run the base station of Prototype1. The only major changes made in firmware were: a)Implementation of the chord map in code memory as a series of ugly if/then statements (since processor data memory was limited), and b)Creation of functions to send PC-AT clock and data signals. The firmware's key functions were:

- **timer0_isr (void)** -- A timer driven interrupt service routine used to count 1 millisecond. (For delays within the main program).
- **timer2_isr (void)** -- A timer driven interrupt service routine used to count .0002 sec--half a PC-AT keyboard clock cycle (bdclk=25KHz).
- **chord_is_changing ()** -- A routine which returns TRUE if the chord is changing.
- **settled ()** -- A routine that returns one when the current chord remains constant for a set ammount of time.
- **auto_repeat ()** -- A routine that allows "typmatic" action: the repeated typing of characters by keeping one chord pressed.
- **translate (byte chord)** -- Maps chords to characters.
- **send_bit ()** -- Sends a bit to the keyboard port.
- **send_byte (byte bytetosend)** -- Sends a byte, encapsulated within an 11 bit frame, to the keyboard port.
- **send_make ()** -- Sends the desired character's "make" code: the byte (or bytes) that correspond to the desired character within the PC-AT keyboard protocol.
- **send_break ()** -- Sends the desired character's "break" code: the byte (or bytes) that correspond to the desired character within the PC-AT keyboard protocol.

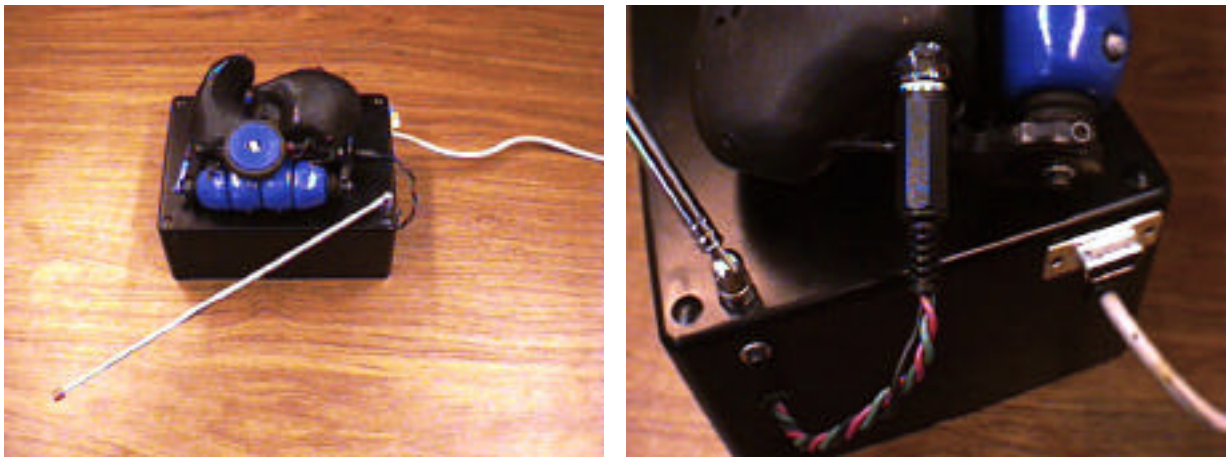
For a complete code listing, see Appendix 12. For diagrams of the PC-AT protocol and the PC keyboard's make/break codes, see Appendix 13.

Initially there was some doubt concerning the feasability of directly implementing a PC-AT port, and indirect methods were considered. One alternative was to buy a "wedge": an RS-232 to PC-AT converter of the sort commonly used for bar-code scanner interfaces. The cost of such a device, however, was prohibitively expensive. Another alternative was to interface the row/column matrix of an existing PC Keyboard. By emulating the signals this matrix would receive from keys, it would be possible to "trick" the keyboard's processor into generating the desired PC-AT signals. This was perhaps the easiest of the alternatives considered, but would have restricted the device's extendibility and seemed like a "cop-out" solution.

Two sources greatly facilitated the development of a direct PC-AT interface. These were an article by Craig Peacock called "Interfacing The PC" and an assembly code sample written by Jim Greene(39 & 40). These sources helped filter out key pieces of information needed to successfully implement the interface such as make/break codes, and how to deal with communications sent from the PC to the keyboard.

Integration

Once the human side and the device sides had been finished, they were joined together. The results are shown below. To the left is the hand unit resting on the base station. To the right is a close up of the hand unit's charging jack.



Finished Hand Unit & Base Station

Problems Encountered During the Development of the Final Prototype

Numerous problems were faced--and many overcome--on the path to a functional final prototype.

Human Side Problems

The human side problems were both physical as well as cognitive. The high-viscosity glue used to create bosses for bolts did not adhere to the thermoformed shell very well. As a result, bosses detached--typically just when the device was being closed and the last bolt was being tightened. This problem was solved through the use of two glue layers--one of thick viscosity to build up the shape of the bosses, the other of great adhesive strength to hold the first glue to the plastic shell.

Another source of difficulty was the problem of finding finger switches with the right characteristics (characteristics described in the "Prototype 1" section). This difficulty came as a complete surprise due to the number and variety of switches available, and the problem has yet to be solved satisfactorily.

The main cognitive problem encountered in the final prototype's human side development was optimization of the chord map. Since the device's primary text entry function and context were not defined, it was difficult to decide whether to optimize for typing speed, learnability, control functions such as navigation, comfort, or some other factor. In the two weeks allotted to chord map creation, a stab was made at optimizing with respect to chord "intuitiveness", however it is yet unclear whether or not this was the best first approach upon which to iterate.

Device Side Problems

A number of difficulties threatened the final prototype's development. A great deal of time was spent designing the hand unit so that it drew no power when no chord buttons were pressed. Theoretically this should have been possible (using the hand unit schematic version 1.5 in Appendix 14), but the design did not work in breadboard tests for reasons that remain unclear. After several days without progress, it was decided that there were concerns of higher priority to address. (All three ICs in the hand unit circuit had low power modes anyway.)

At one point, the project's electronic development was threatened by difficulties encountered while trying to implement a signal delay. Initially it was thought that a 555 timer circuit would work, however inspection of this IC's data sheet revealed that it would not suffice. Next, a circuit was designed using buffer, low pass filter and comparator blocks. When this circuit was tested on a breadboard using run-of-the-mill 711 op-amps for the buffer and comparator blocks, it did not function as planned. After investigation, it was discovered that rail-to-rail op-amps were required due to the tight supply

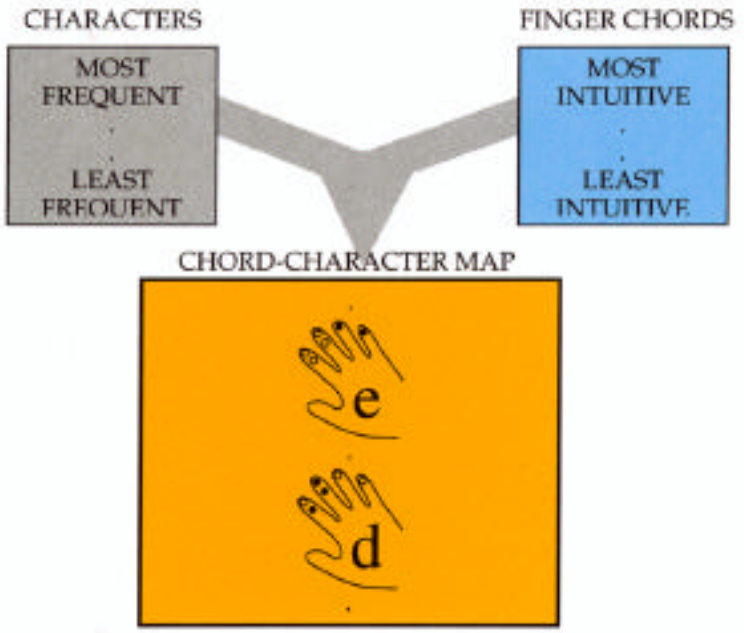
voltage range (0V-5V). When rail-to-rail op-amps were used, the delay functioned properly.

Another difficulty encountered during device side development was the inability to properly actuate "extended" characters corresponding to make codes composed of more than one byte. This prevented characters such as <delete> and navigation signals like <up-arrow> from working on the new device. For four weeks, it was thought that the problem's root was an incomplete understanding of the transmission protocol for "extended" characters. The difficulty turned out to be a basic mistake in a branch statement of the firmware's main subroutine. In this situation, overlooking the obvious caused the loss of precious time.

Developing A Chord-to-Character Map

After designing the human side and device side of the interface there remained one additional challenge of great complexity: the creation of a chord-to-character mapping scheme.

A number of possible approaches were considered. One approach--used to create the chording system for the "Data Egg" (a device described in the "State of the Art" section)--was have chords mapped to characters based characters' shapes. In a system like this, the character "." might be actuated by pressing one button while an "I" might be actuated by pressing a line of buttons. Another approach considered was to create a chord map based on characters' ASCII numbers. Finger chords in this system would in effect be the ASCII numbers in base two. The approach finally decided upon was to create a hierarchy of chords from most intuitive to "least intuitive", and match it with a hierarchy of characters ranked from most frequent to least frequent.

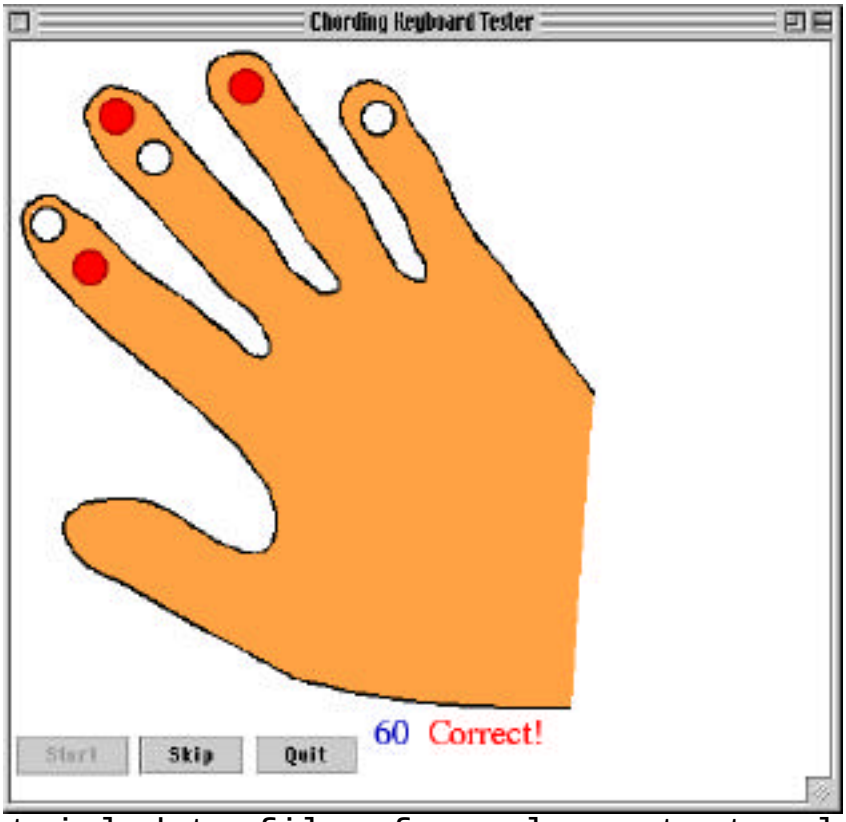


Developing a Hierarchy of Chord Intuitiveness

A hierarchy of chord intuitiveness was developed through the aid of a computer program created by software engineer Michael Fromberger. This program displayed all 63 possible chord combinations in random order, and prompted test subjects to "echo" each chord by pressing the buttons on the hand unit. Each chord remained on display until it had either been successfully actuated or else skipped. The program logged the following data for each chord:

- **Time:** The time taken, in milliseconds, to press (or skip) the chord.
- **Done**: A binary value; 1 if the chord was successfully pressed, 0 if the chord was skipped.
- **Errors**: The number of errors made before a chord was completed or skipped.

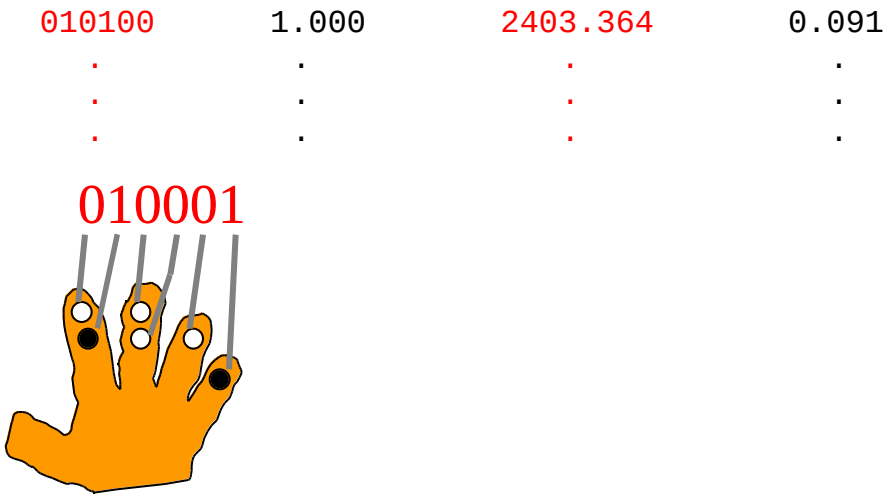
A sample of the program display appears below, and its corresponding code listings appear in Appendix 15.



The trial data files from eleven test volunteers were sorted by chord, then averaged. The resulting "average" data file was sorted by the time taken to complete or skip chords. This was done based on the assumption that more intuitive chords would be pressed more quickly than less intuitive chords. The code used to carry out these steps--together with resulting chord hierarchy--can be found in Appendix 16. A segment of the chord hierarchy appears below.

Time-Sorted "Average" Data File:

FingerChord	Done(0-1)	Time(msec)	Errors
000100	1.000	1442.182	0.000
000001	1.000	1449.455	0.000
000010	1.000	1608.000	0.000
101000	1.000	1616.000	0.000
001000	1.000	1743.455	0.000
001100	0.909	1868.182	0.545
011000	1.000	1967.455	0.000
000110	1.000	2115.545	0.091
000011	1.000	2116.727	0.182
001010	0.909	2205.909	0.273
010000	1.000	2398.091	0.000



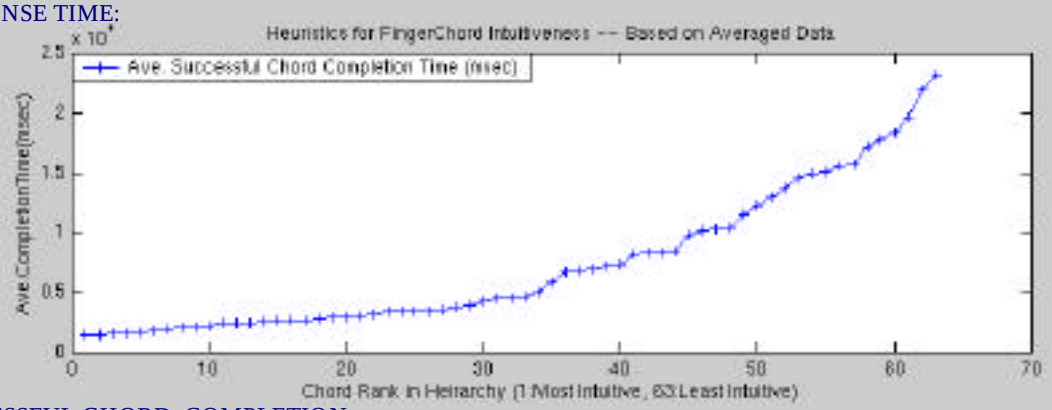
The decision to sort chords based on the "time" heuristic was somewhat arbitrary. Chords could have been sorted by either of the other two intuitiveness heuristics: "done" or "errors". Chords also could have been sorted based on a function of all three heuristics.

The three heuristics provide roughly the same information; this is evident from an inspection of the graphs on the following page. On these graphs, the x-axis corresponds to time-based chord hierarchy values ranging from 1 ("most intuitive") to 63 ("least intuitive"). The y-axes of the three graphs correspond to the three intuitiveness heuristics that were recorded. The fact that the rise in time taken to complete or skip chords is matched by a general decline in chord completion and a rise in the number of errors made before completing or skipping a chord suggest that the time-based hierarchy is an accurate measure of chord intuitiveness. The data for the "errors" and "done" heuristics support the time-based

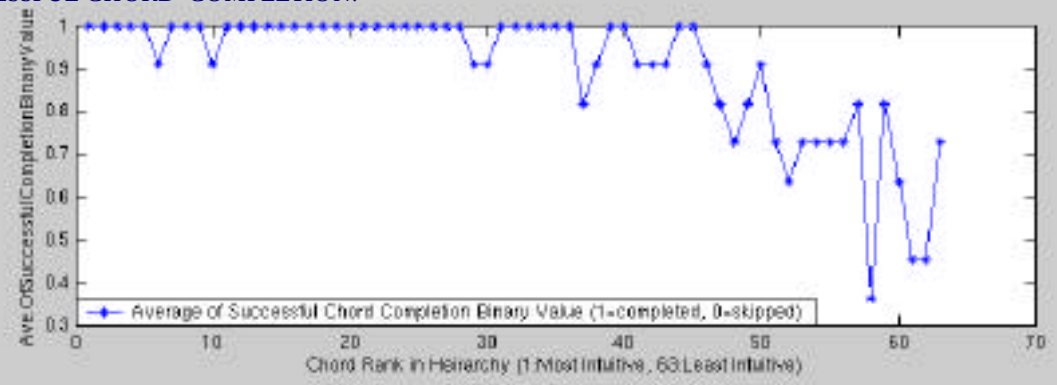
hierarchy for most chords. (The code used to generate the graphs appears in Appendix 16).

Alternative Heuristics for Determining a Hierarchy of "Most Intuitive" Finger Chords

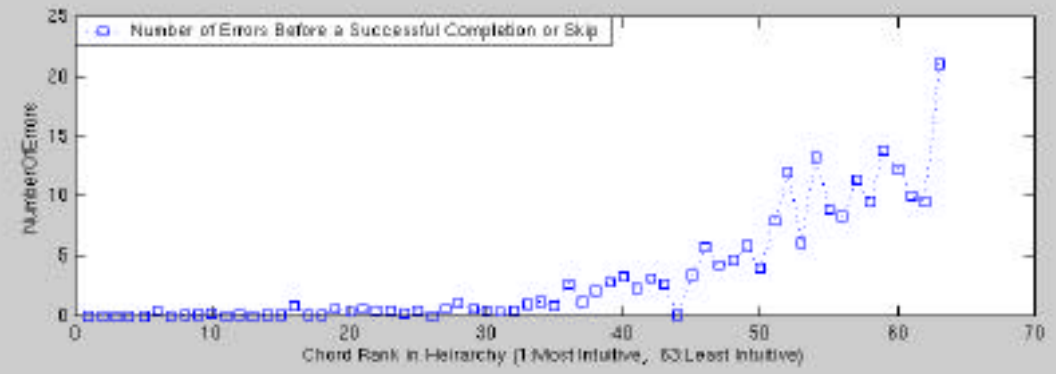
RESPONSE TIME:



SUCCESSFUL CHORD COMPLETION:



NUMBER OF ERRORS:



Developing a Hierarchy of Character Frequency

In order to develop a hierarchy of character frequencies, the work of various linguists was consulted. Several were able to share hierarchies of English letters such as the following:

MorseCode: E T A I N O S H R D L U C M F W Y G P B V K Q J X Z
Journals: E T A O N I S R H L D C M U F P G W Y B V K J X Q Z
Religious Texts: E T I A O N S R H L D C U M F P Y W G B V K X J Q Z
Science Texts: E T A I O N S R H L C D U M F P G Y B W V K X Q J Z
General Fiction: E T A O H N I S R D L U W M C G F Y P B K V J X Z Q
Mix of the Above: E T A O I N S R H L D C U M F P G W Y B V K X J Q Z

supplied by Steven Schaufele, a linguistics professor at the University of Soochow in Taiwan(41).

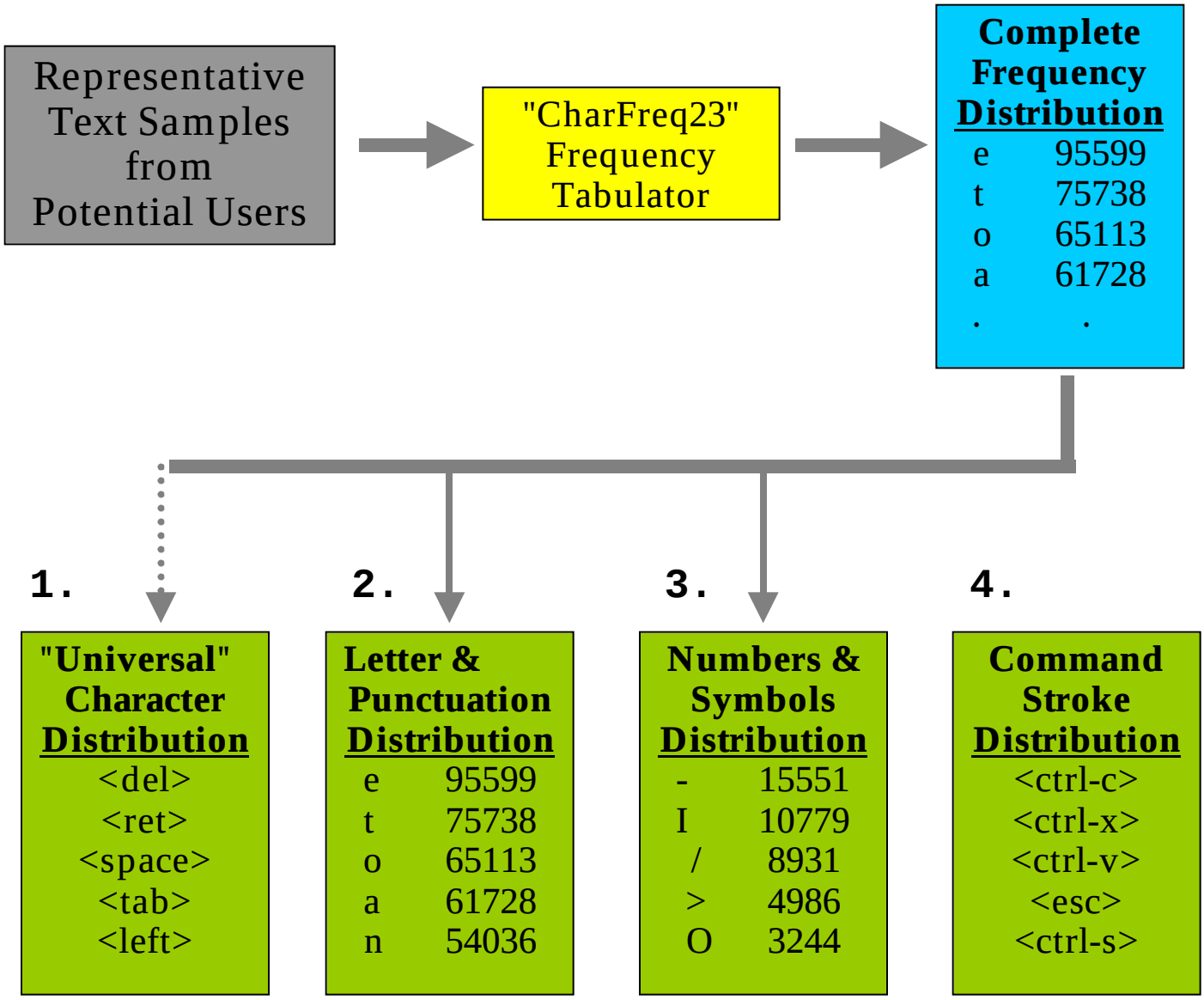
These hierarchies raise an important question: What dialects of English would potential users use?

Although the hierarchies shown above provide useful data concerning letters they do not include numbers, punctuation, carriage returns, spaces, and numerous other symbols that are often used while typing.

Since character frequency depends on the specific language (or dialect) typed and none of the character frequencies found through research included non-alphabetic characters, the decision was made to develop a new character hierarchy, based on representative sample texts submitted by potential users. A chart illustrating the process used to create this hierarchy appears on the following page.

Sample texts were lumped together in one gargantuan text file, and fed to a freeware frequency tabulating program called "Chrfreq2.3" developed by J R Ferguson(35). The output of this program was a complete frequency-based hierarchy of the ASCII character set. This hierarchy was then divided into three bins: "Universal" characters (characters that

Developing a Frequency-Based Character Heirarchy



would be necessary in most text entry situations) "Letters & Punctuation" (the frequencies of lower-case letters were used because lower case is more common), and "Numbers & Symbols". A fourth bin was created to hold commonly used text entry commands such as copy, cut & paste. The hierarchy in this fourth bin was arbitrary, since command strokes did not appear in the plain text samples.

Characters were assigned to finger chords in the following manner: First, the most intuitive chords across each of the device's four "cases" (Lower,Upper, Math&Symbol and Control) were matched to the "Universal" character distribution. Next, the most intuitive chords yet unassigned in the Lower and Upper cases were mapped to the "Letters and Punctuation" distribution. After doing this, the most intuitive chords yet unassigned in the Math&Symbol case were matched to the Numbers&Symbols distribution. Finally, control commands were mapped to unassigned chords in the Control case based on their character makeup--for example "cut", <control-x> was mapped to the same chord as "x" in the Lower case (which was the same chord as "X' in the Upper case). The resulting chord map appears in table format on the following page.

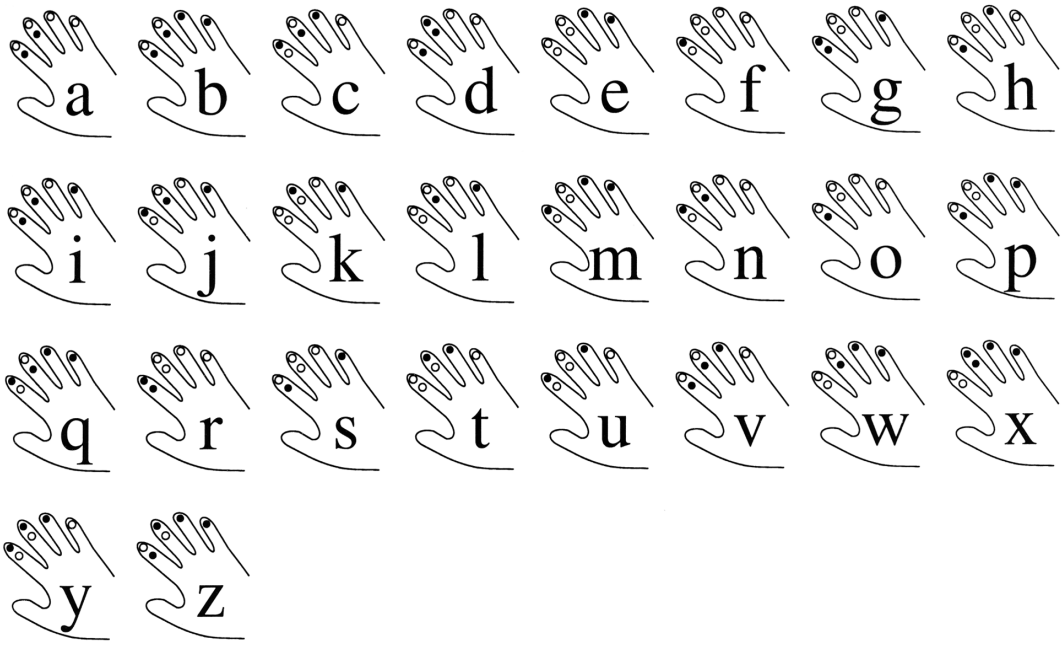
This chord map is somewhat dense to read, and the 6-bit binary numbers it uses to represent finger chords are cryptic. To make the chord map more easily learnable for potential users, a more visual chord map was created for each of the four cases. These maps show characters inside hand images with symbols for pressed and released buttons. The visual chord maps were generated through automation; a Perl script written by Michael Fromberger was used to create post-script image files--one for every character in the chord map. Automated image generation produced clear, crisp images that could be changed quickly and easily during chord map refinement. The visual chord maps appear on the pages following the table chord map, and related code listings appear in Appendix 16.

The Final Chord Map In Condensed Table Format

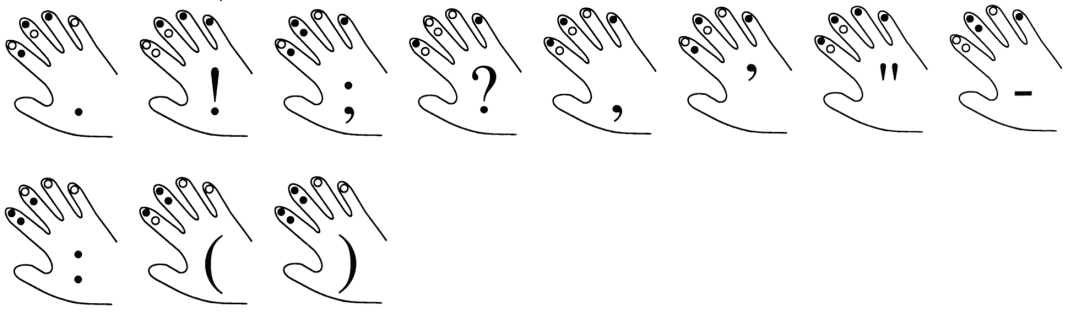
	Hand Byte Chord			MATH &	
Position	1=pressed,0=released	LOWER	UPPER	SYMBOL	CONTROL & OTHER
1	010100	a	A	@	<ALT>
2	010111	b	B	<	<CTL-ALT-DEL>
3	100110	c	C	_	<CTRL-C>
4	011100	d	D	3	
5	000011	e	E	>	<ESC>
6	100000	f	F	1	
7	110001	g	G	7	
8	010010	h	H	5	
9	010101	I	I	|	
10	100101	j	J	{	
11	001001	k	K	[	
12	000101	l	L	4	
13	100011	m	M	8	
14	100100	n	N	/	<CTRL-N>
15	010000	o	O	O	<CTRL-O>
16	010011	p	P	+	<CTRL-P>
17	100111	q	Q	^	<CTRL-Q>
18	110000	r	R	2	
19	010001	s	S	$	<CTRL-S>
20	001010	t	T	=	
21	100010	u	U	9	
22	010110	v	V	]	<CTRL-V>
23	000111	w	W	*	<CTRL-W>
24	001111	x	X	%	<CTRL-X>
25	101010	y	Y	6	
26	011011	z	Z	#	
27	001101	-	-	-	
28	011010	.	.	~	
29	101001	,	,	`	
30	100001	?	?		
31	001011	!	!	\	
32	110100	:	:		
33	011101	;	;		
34	101011	"	"	}	
35	011001	'	'	&	<"and">
36	101100	(	(	(	
37	111100	)	)	)	
38	101000	<TAB>	<TAB>	<TAB>	<SHFT-TAB>
39	000001	<RETURN>	<RETURN>	<RETURN>	<RETURN>
40	000010	<SPACE>	<SPACE>	<SPACE>	<SPACE>
41	000100	<DELETE>	<DELETE>	<DELETE>	<DELETE>
42					
43	001000	<LEFT>	<LEFT>	<LEFT>	<SHFT-LEFT>
44	001100	<RIGHT>	<RIGHT>	<RIGHT>	<SHFT-RIGHT>
45	011000	<UP>	<UP>	<UP>	<SHFT-UP>
46	000110	<DOWN>	<DOWN>	<DOWN>	<SHFT-DOWN>

Lower Case Finger Chords

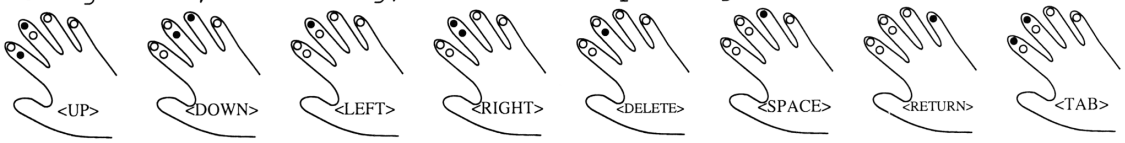
Letters:



Punctuation, Parenthesis & Quotes:

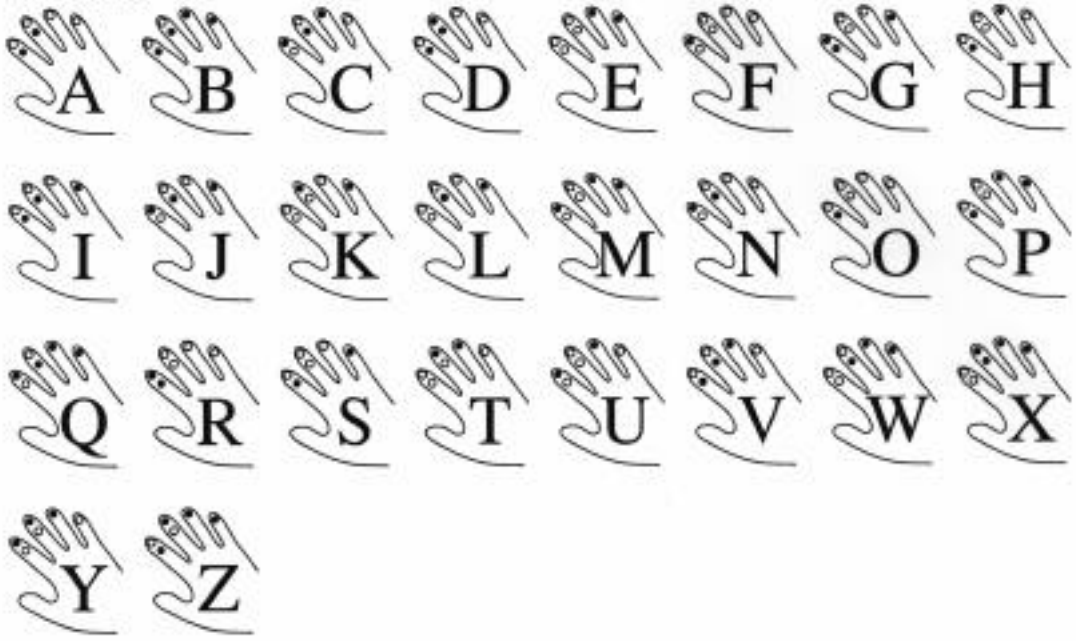


Navigation, Deleting, Return & Spacing:



Upper Case Finger Chords

Letters:



Punctuation, Parenthesis & Quotes:

Navigation, Deleting, Return & Spacing:

Math & Symbol Finger Chords

Numbers:

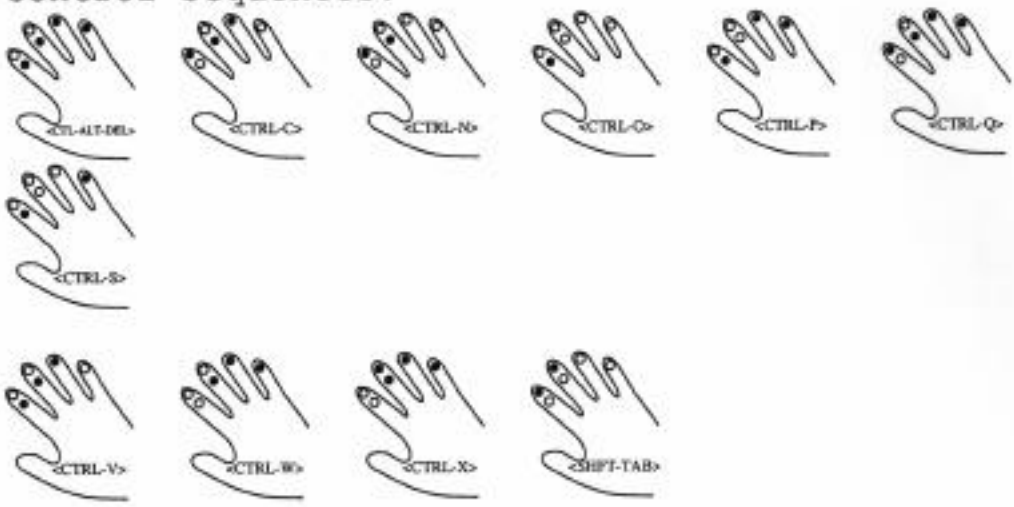
Operands:

Deletion, Spacing & Navigation:

Assorted Symbol Characters:

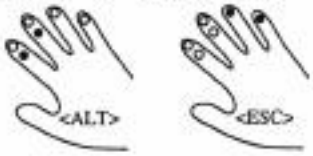
Control Finger Chords

Control Sequences:

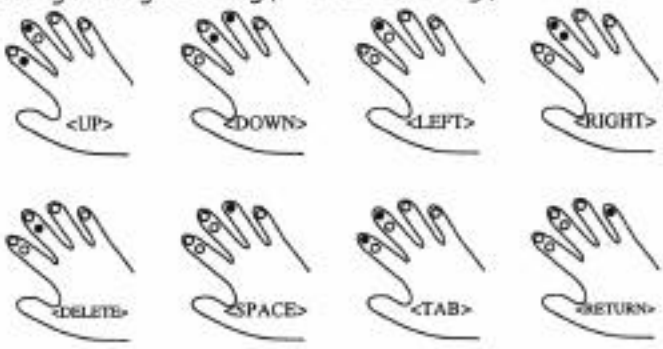


Words:

Alt & Escape:



Highlighting, Deleting, Return & Spacing:



Formal Evaluation of the Final Prototype

The final prototype was evaluated formally according to the criteria presented in the "Evaluation Methodology" section of this paper. The results appear in table form on the following three pages. Below are comments on some of the tests and their results.

Weight and size were easy to judge through use of a ruler, balance and graduated cylinder. Volume was measured through water displacement. The results were: Weight=220g, MaximalDimension=15cm & Volume=$(8.43cm)^3$.

Some durability tests were conducted, others were not. The hand unit operated after 5 minutes in a .5°c tub of ice water and 5 minutes in a 45°c bath (protected in both cases by a plastic bag). It failed the 1m "drop" test miserably; glue joints broke in two places and a soldered lead came loose, short-circuiting and destroying a chip. This failure alerted the developer to the reality that the device was a prototype--not a product. It was resolved that no more potentially catastrophic tests would be run before the thesis presentation. Needless to say, the "shower" test was not conducted.

Typing Duration was tested by "hard wiring" the finger chord corresponding to <enter> then opening up a dialog box in the Microsoft "Notepad" application which provided visual feedback confirming that <enter> was being pressed repeatedly. After 6 hours, the dialog box continued to provide affirmative feedback.

Though there was not time to obtain a complete cost estimate for a product based on the final prototype, several estimates were obtained: 1$/pc board (research engineer Doug Fraser), $25,000 for the shape mold (machine shop supervisor Kevin Barron), 10¢/unit for plastic (CAD/CAM supervisor Pete Fontaine) and 15¢/unit for nuts & bolts (Kevin Barron).

Formal Evaluation

SPECIFICATION	EVALUATION TESTS	
PORTABILITY		
· Weight	✓ < 150g (weighed)	+ 1
· Size	✓ <16cm max. dimension (measured)	+.3
	✓ <$(14cm)^3$ (measured)	+.3
	✓ Fits in handbag & large coat pocket	+.4
· Durability	❑ Survives 1 minute of operation in a shower	+.3
	❑ Survives a 1m drop onto a hard concrete floor	+.4
	✓ Survives operation at 0^0C and at 37^0C	+.3
· Range	✓ Operates @ 2m distance from base station	+ 1
· Typing Time	✓ Sends text for six hours continuously on one battery charge	+ 1
	4.3 out of 5(max)	
HEALTH & SAFETY		
· Ergonomic Correctness	❑ Hand maintains a comfortable resting position w/o pain	+.5
	✓ Neutral wrist position	+.5
	✓ Doesn't demand static posture	+.5
	✓ Doesn't demand excessive digit motion or force	+.5
		+.5
· Physical Safety	✓ Does not restrict body motion	+.5
	✓ No sharp edges	+.4
	✓ Toxification not a risk	+.3
	✓ Meets FCC specs for radiation safety	+.3
	4.5 out of 5(max)	

Formal Evaluation (Continued)

SPECIFICATION	EVALUATION
ACCEPTABILITY	❑Wins head to head against other designs in the same survey group of potential users as "most acceptable". +5 **0** out of 5(max) (Test not yet conducted)
EFFECTIVENESS · Learnability · Speed	❑Possible to learn typing system at a basic level in < 5 hours. +3 ❑15 wpm average speed possible after 10 hrs practice. +1 ❑30 wpm average speed possible after 20 hrs practice. +1 **0** out of 5(max) (Test not yet conducted)
MULTITASK-ABILITY	While typing, ✓Can multitask vocally +1.5 ✓Can multitask with one hand +1 ❑Can multitask with other hand +1 ✓Can multitask with body +1.5 **4** out of 5(max)
FEASABILITY	✓Two prototypes developed within real world constraints +5 **5** out of 5(max)

Formal Evaluation (Continued)

SPECIFICATION	**EVALUATION TEST**
COST	✓ Development Cost < $600US +2 ❑ Estimated cost of a product based on final prototype doesn't exceed $100US, given a hypothetical production run of 100,000 units. (Cost estimates given by a panel of experts). +3 _____ **2** out of 5(max) (Second test not yet conducted)

Informal Evaluation of the Final Prototype

In addition to the completed tests of the formal evaluation, a learnability & speed test was initiated and a final survey conducted.

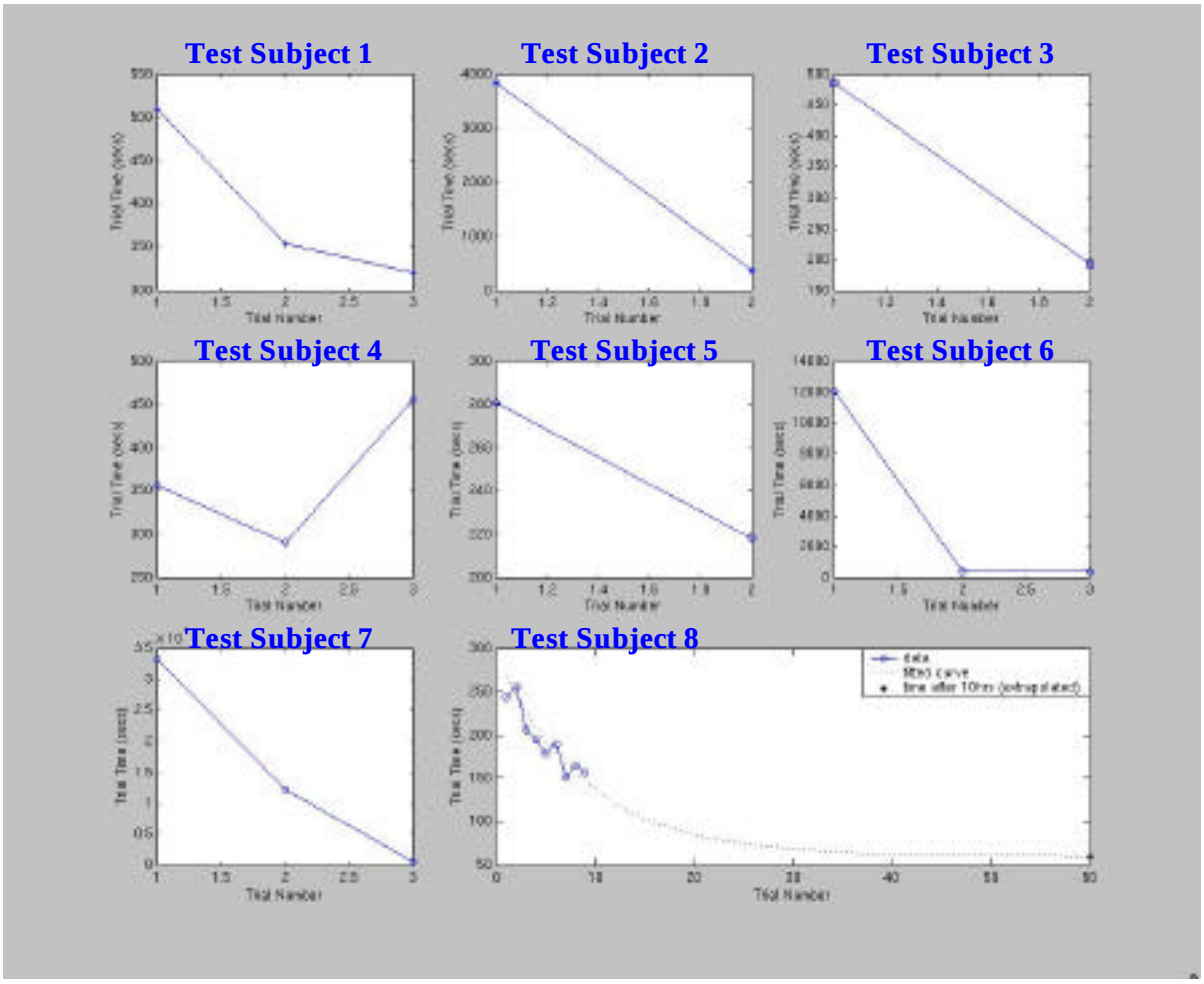
Learnability & Speed Testing

The set of all characters was too large to test effectively given the time constraints, so it was decided to test a reduced character set: the set of numbers and math operators. This set was chosen because it was a small set that--once learned--could serve a purpose: operation of a software calculator. New techniques are easier to learn with a purpose in mind, and the ability to operate a software calculator provided a minor degree of motivation (as did the Oreo® cookies provided.)

To test the reduced character set's learnability, test subjects ran a computer program. This program prompted for 30 random numbers and math operators, and timed how long it took test subjects to correctly enter these numbers and math operators via the hand-held unit. Test subjects took the test multiple times, and their times were logged in a file. Sample output from the program is shown below, results appear in graph format on the following page, and relevant code listings are included in Appendix 17.

```
The "Calculator" Test
      Press <return> to begin:
      1. <<96>>96
      2. <<+>>+
      3. <<24>>2
      Please type <<24>>.
      3. <<24>>24
      .
      .
      29. <</>>/
      30. <<90>>90
      Thanks for taking the time to do this trial!
      Your time this time round was: 70.351 seconds.
      Your trial times (in seconds) are: [1200.351   900.342]
```

"Calculator" Time Trial Test Results



Unfortunately in most cases test subjects took the test too few times for accurate learning curves to be fitted to the data. For the most part, there was dramatic improvement between the first and second trial, showing that learning was taking place. One test subject took the test numerous times, and a curve was fitted to this subject's trial data, using a function of the form:

$$Ae^{-Bt} + C$$

A, *B* and *C* were chosen through trial and error, and the curve fit assessed visually for lack of knowledge of better curve fitting techniques.

The curve fitted to Test Subject 8's trial data was then used to extrapolate an estimation for typing speed after 10 hours of practice. This estimation relied on the following assumptions.

- Each "Calculator" test trial takes 10 minutes, so 60 trials are possible in 10 hours.
- 90 characters are typed in each trial.
- On average, there are 5 characters in a word(42).

Based on these assumptions and the fitted curve data, the following calculation was made:

$$\text{Predicted WPM After 10 hours} == \frac{\text{characters/finaltrial} * 60\text{sec/min}}{\text{seconds/finaltrial} * 5\text{chars/word}}$$

$$== \frac{90*60}{60.3197*5} == 17.9066 \cong \mathbf{18\ WPM}$$

18 words per minute after 10 hours is three WPM faster than the goal set in the "Specifications" section of 15 words per minute after 10 hours.

Due to the fewness of data points, the dubious curve fitting procedure and the numerous assumptions made, great faith should not be place in the 18 WPM figure, however it does serve as an order of magnitude estimate.

The Final Prototype Survey

After completing the "Calculator" time trial test, volunteers were asked to complete one final survey regarding the interface. Survey and compiled results are in Appendix 18, and the results are summarized below:

Number of responses: (8)

Occupation:

Engineering Student:	(4)
Computer Science Major:	(1)
Geology Student:	(1)
Woodshop Instructor:	(2)

Interface "Pros":
Comfortable, mobile, allows free range of body motion, requires only one hand, is of good size & weight.

Interface Cons:
Some chords awkward or impossible, difficult to learn all chords, wires exposed, inadequate button size and tactile feedback, grip needs padding, and it is difficult to type fast.

Could Envision Typing With the Device: Yes(100%)

Envisioned Uses for the Device:
Multi-purpose controller, mobile PDA or computer keyboard, extensor robotics controller, web navigation, tool for handicapped access.

Want the chord map to be programmable: Yes(100%)

Overall Acceptability Rating [1=LO 5=HI] taking into account aesthetic, sociological, ergonomic and cognititve factors: Average: 4.1

The survey indicated that there were problems that needed to be addressed regarding the actuators, grip padding and the chord map. People generally found the device acceptable, felt that it could be of use in typing and controlling contexts, and liked the free range of motion and mobility it made possible.

Future Work

The interface can be improved in numerous ways, and still needs to be properly tested on numerous fronts. Appropriate goals for future work include the following actions.

Human Side:

- **Add Tactile symbols** to the case selector thumb wheel.
- **Improve the finger buttons**. Better tactile feedback and larger touch surfaces are required.
- **Add padding** to grip area.
- **Reduce external wiring,** possibly through "signal superposition" switches. (Have all switches on one circuit line, and have each contribute a distinctive signal when pressed. Each switches contribution to the signal can then be decoded by the circuit inside the hand-unit's body--this idea was proposed by research engineer Doug Fraser.)
- **Refine the chord map and or develop different maps.** Based on informal testing, the device seems to be well suited to tasks which would require a limited (and thus more learnable) chord set--tasks such as web navigation or control of household devices. Given this observation, it might make sense to optimize for controlling or navigation functions in future chord-mapping efforts.

Device Side:

- **Encrypt wireless data** transmission.
- **Add programmability** to chord map so that the user can easily re-map all (or some) of the chords to characters, commands and strings. Adding this feature early on might provide insight into the

device's most suitable functions and methods for chord map optimization.

- **Simplify** base station hardware -- use microprocessor software to do the work of the decoder and inverters that are currently necessary to the base station circuit.
- **Reduce latency** inherent in the wireless link. This could be accomplished simply by speeding up the relatively slow wireless data transfer rate.
- **Improve battery charging** circuitry.
- **Reduce standby power requirements** of the hand unit circuitry.

Testing:

- **Learnability, Usability and Acceptability** testing are most important!!! If people can't learn to use the interface or don't want to do so, its not a useful tool. The reactions that children and elderly folks have toward the interface have yet to be witnessed.
- **Comfort**, **Water Resistance, Shock Resistance**.

Resources & Acknowledgment

This project was made possible through the assistance and expertise of a number of people, institutions, and written works. I've tried to remember and include as many of these contributors as possible below.

People and Institutions

· David Stratton
· Clayton Okino
· Doug Fraser
· Michael Fromberger
· Lisa Tiraboschi
· Ted Cooley
· Stu Trembley
· Francis Kennedy
· Charlie Sullivan
· Pete Fontaine
· Leonard Parker
· John Collier
· Kevin Baron
· Michael Ibey
· Rene & Roger Dauphinais
· The many people who took the time to answer surveys and offer suggestions.
· Cherry Corp. Electronics
· Glolab.com RF Electronics

Material/Environmental Resources

· Thayer School Machine Shop
· Thayer School Digital & Analog Labs
· Dartmouth College
· Mom and Dad

Works Cited & Consulted

1. Gleick, James. Faster. Pantheon Books, NY, 1999.

2. Beeching, Wilfred A. Century of the TypeWriter. St. Martin's Press, NY, 1974.

3. Herkimer County Historical Society. The History of the TypeWriter. Herkimer, NY, 1923.

4. Kittler, Friedrich A. Gramophone, Film, TypeWriter. Stanford University Press, Stanford, 1986.

5. http://www.cs.ucl.ac.uk/staff/b.rosenberg/kbd/

6. Kelley, Whitmore. Interview concerning mind-controlled computer interface systems. Dartmouth College, 2000.

7. Hailes, Katherine N. How We Became PostHuman. University of Cicago Press, Chicago, 1999.

8. Goldberg, Beverly. Overcoming High Tech Anxiety. Jossey-Bass, San Francisco, 1999.

9. Denning, Peter ed. Talking Back to the Machine. Copernicus, NY, 1999.

10. Cooper, Alan. The Inmates are Running the Assylum. Alan Cooper, 1999.

11. Orr, Linda V. Computer Anxiety. University of Southern Maine, http://www.usm.maine.edu/~wm/lindap~1.html.

12. Ruegg, David. Repetitive Stress Injury: A Handbook of Preservation & Recovery. Largo, http://home.clara.net./ruegg/info.htm.

13. Snead, Elizabeth. "Virtual Certainty: Films Reflect Computer Fears". USA Today. April 21 1999, 1D.

14. Harmon, Amy. "Talk, Type, Read EMail". New York Times. July 23 1998, Late Edition, G1.

15. Price Waterhouse Coopers, Technology Forecast, PWC, Menlo Park CA, 1998.

16. Electronic Industries Alliance. Electronic Market Data Book. EIA, Arlington, 1999.

17. OSHA, "Secretaries Day Memorandum". http://www.osha.gov/media/secretary97/memo.html.

18. Smith, MJ, et al. "Occupational Stress in Human Computer Interaction" Ind Health, April 1999, 157-173.

19. Turner, Rob. "Making IT Personal" Money. May, 1998, 155-157.

20. Ferguson, Owen. "A Look into the Future". Computer Dealer News. December 10 1999, 1-53.

21. Chadderdon, Lisa. "Eighty-Sixing the Nine-to Five". Architecture. December 1999, 92-95.

22. Emmerson, Bob. "The Magic of Mobile". Communications International. November 1999, 64.

23. Hudson, Marion E. "Hoteling: Offices a la Carte". Office Systems. September 1999, 28-34.

24. O'Brien, Jennifer. "IBM/Psion Development Alliance Strives to Pump Up PDA Features". Computer Dealer News, August 27 1999, 9.

25. Thomas, Richard. "The World is Your Office". Management Today. July 1999, 78-84.

26. Charles, Kirk. "Jack or Jill of All Trades". Black Enterprise. July 1998, 154.

27. Graps, Amara. "Ergonomic Computing (or Dont Let Your Computer Cripple You!)". http://www.amara.com/aboutme/rsi.html.

28. Dignan, Larry V. "iMac Smashes ComUSA's Sales Records". http://www.zdnet.com/zdnn/stories/news/0,4586,2164081,00.htm

29. Liberty Mutual. "Office Ergonomics" pamphlet series. Krames Communications, 1996.

30. Sharpe, Rochelle. "Work Week." The Wallstreet Journal. April 9th, 1996.

31. Rodgers, Suzanne ed. Ergonomic Design for People at Work. Eastman Kodak Company Ergonomics Group Health and Environmental Laboratory. Van Nostrand Reinhold, NY, 1986.

32. Chording Keyboards, Typing Injury FAQ. http://www.tifaq.org/keyboards/chording-keyboards.html#Infogrip

33. NIOSH, Carpel Tunnel Syndrome. http://www.cdc.gov/niosh/ctsfs.html

34. Interview with Denise Finch, Ocupational Therapist. Dartmouth College, March 8, 2000.

35. J R Ferguson, CHRFREQ (Share-Ware MS DOS Character frequency tabulating application), http://hello.to/ferguson.

36. Adam D. Tinniswood, et al., "Computations of SAR Distributions for Two Anatomically Based Models of the Human Head Using CAD Files of Commercial Telephones and the Parallelized FDTD Code", IEEE Transactions On Antennas and Propagation, Vol. 46, No. 6, June 1998, p829.

37. Ralph Tenny. "Nicad Battery Charger". http://www.dprg.org/nicad_charger.html June 4, 2000

38. Thomas J. Armstrong. (Case study on hand size for the class: Occupational Ergonomics Fall 99), http://www.engin.umich.edu/class/ioe433/Biomechanics/Handsurvey.html June 4, 2000

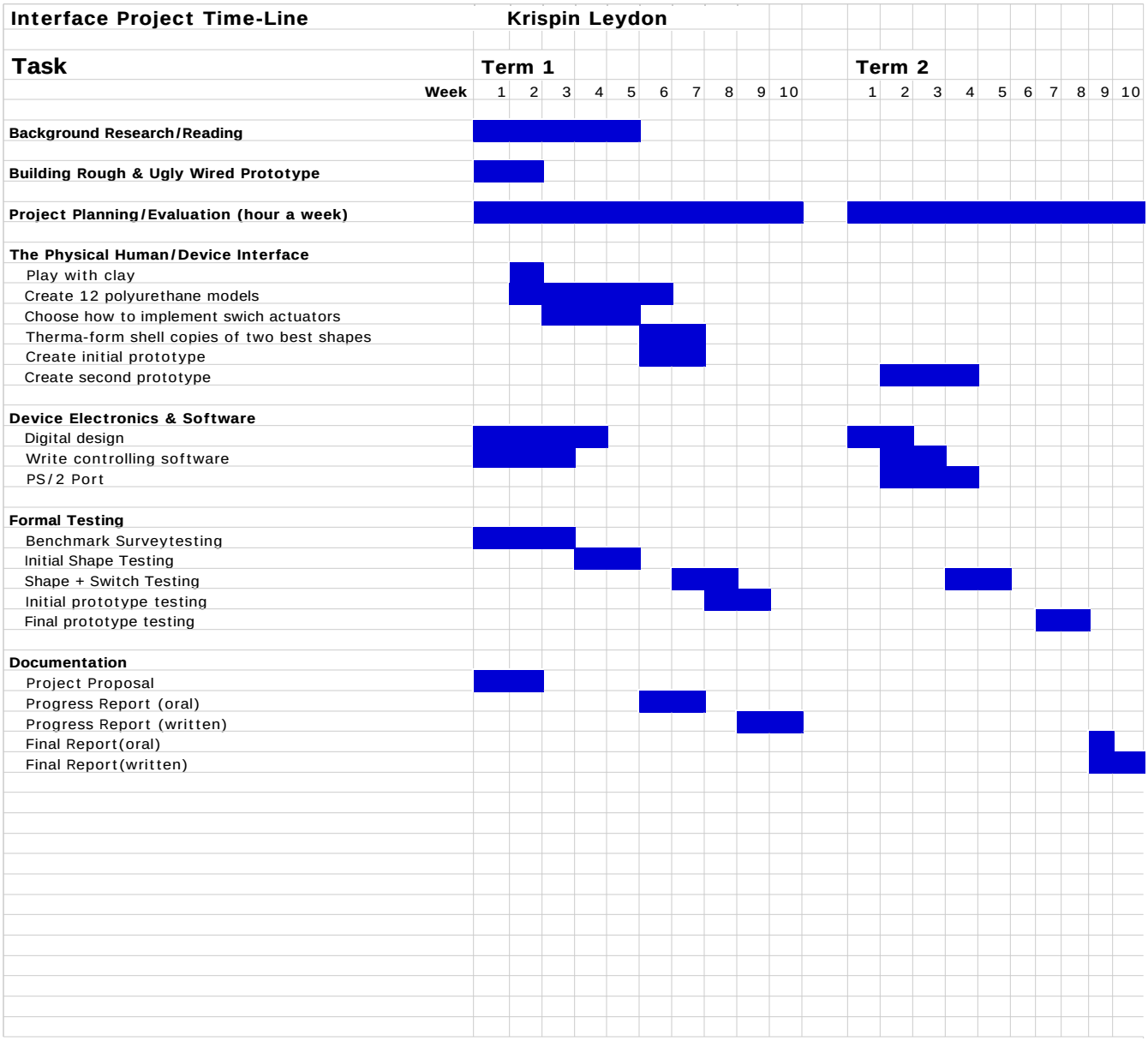
39. Craig Peacock. "Interfacing the PC Keyboard", http://www.beyondlogic.org/ Jan 21, 2000.

40. Jim Green. "Keyboard Scanner Using the 83C751". (An assembly code sample) March 27, 2000.

41. Steven Schaufele. (Answer to a web bulletin posting concerning character frequencies), English Department, Soochow University, Waishuanghsi Campus, Taipei 11102, Taiwan, ROC, fcosw5@mbm1.scu.edu.tw

42. Noah Hearle. <http://ds.dial.pipex.com/town/park/yfn77/Academic/Maths/Sentence.html>

Revised Time Line



Keyboard History & "Herstory"

The dominant interface device used by humans to control machines over the past two centuries has been the typewriter keyboard . This interface "completely revolutionized the modern business world" implemented as the typewriter, then did so again as the interface of choice for the personal computer(2, 33). The fact that this interface has both initiated revolutions as well as survived two centuries of the most rapid technological growth humanity has ever experienced testifies to the interface's effectiveness as a basic solution to the problem of moving text information from man to machine.

The QWERTY interface was developed by Christopher Latham Sholes working for the Remington company. The invention of the typewriter was the work of many hands over an extended period of time; Sholes was the52nd person to "invent" the typewriter(2, 28).

The development of the typewriter--like so many works developed *using* the typewriter --is a story steeped in irony. Despite the machine's revolutionary nature, it "attracted very little attention when...first exhibited"--generally less than the (female) typist operating it(2, 33). (The term "typewriter" originally applied to both typist and the device, and led "to predictable Music Hall gags about men working with typewriters on their knees")(2, 34). Ironically, the women typists who made chauvenistic humor possible were spearheading the women's professional movement in the United States.

Perhaps the greatest irony associated with the development of the typewriter is the fact that it emerged as a success. The interface was designed for machines first, and humans second. The "reason for the QWERTY layout was to space frequent digrams [two letter combinations] further away from each other, reducing the number of [mechanical] jams"(5). The QWERTY layout was not designed to slow people down,

contrary to poular belief, however its design did not optimize typing speed or usability either.

The QWERTY--or "Universal"--keyboard became a natural interface choice for mainframes and personal computers. It allowed people to interface a complex, new and potentially intimidating machine through a layer of simplifying abstraction that was known and familiar. In this way, the Universal keyboard interface has served as a bridge, allowing ordinary people to cross between old and new worlds with a minimum of hardship.

Psychological Considerations in Computer Interface Design

> ***Our writing tools are also working on our thoughts.***
>
> *--Nietzsche, 1881*

Nietzsche, one of the first people to rely on a writing machine, was also an early contributor to the dialogue concerning the subtle influence machines exert over their creators(4, 200). His concern was echoed by Konrad Zuse half a century later, when Zuse implemented the first calculating machine capable of conditonal jumps.

> ***I was, [Zuse said,] nervous about taking that step [implementing IF THEN command sequences]. As long as that wire had not been laid, computers can be easily overseen and controlled in their possibilities and effects. But once unrestricted... processing becomes a possibility, it is difficult to recognize the point at which one could say: up to this point, but no further(4, 258).***

The anxiety expressed by Nietzsche and Zuse is one of the most consistent themes in science fiction. It is present in classic works such as Arthur C. Clark's 2001 A Space Oddesy and Issac Assimov's Robots series, as well as in contemporary works such as The Matrix and Virtuosity. Notes film historian Leonard Maltin, "We are uncomfortable with what computers represent: the loss of control of our lives. We want computers to help us, not rule us. They are supposed to be a tool, a slave, not our masters"(13). Josef Rusnak, director of the sci-fi thriller The 13th Floor remarks that though "the computer...has become part of our daily existence...its control is terrifying"(13).

Fears concerning computer control influence consumer choices; people tend not to buy things they perceive to be a threatening or untrustworthy. Consider the fact that the only new computer input device to gain near complete market penetration over the course of the last thirty years is a small, non-threatening object associated with an endearing, harmless rodent (the "mouse"). In addition, the best selling personal computer in recent history is a machine designed expressly NOT to look like a computer (Apple's "iMac")(28). According to Pattie Maes, associate professor at the MIT Media Lab, "People will adopt [new digital] devices only if they can really trust [them]...humans always have to feel in control--you don't want [a] computer running your life"(9).

The fear of being replaced or controlled by computers is not just material for historical footnotes, fiction, and market predictions; it is pervasive force affecting people's physical and psychological well-being. According to Linda Orr of the University of Southern Maine, "Feelings of anxiety toward computers and computer use is common, affecting 30 to 40 per cent of the [U.S.] population"(11, 2). According to a study done by the department of industrial engineering of the University of Wisconsin, "the effects of the stress of human computer interaction in the workplace are...somatic complaints...anxiety, fear and anger"(18). Alan Cooper, author of the treatise on interface design *The Inmates are Running the Assylum*, notes that widespread use of computers in the workplace has led to a Tourette's-like response: "[You] can walk down the halls of most office buildings and hear otherwise normal people sitting in front of their monitors, jaws clenched, swearing repeatedly in a rictus of fury"(10). The trend borders on humerous now, but its extension into the future may not be so funny. Konrad Zuse's question returns to haunt: When do you draw the line?

Computer anxiety is of crucial importance in the design of hand operated textual input devices, for

the hand--together with language--are two defining characteristics of human beings. Martin Heidegger in his essay On the Hand and TypeWriter goes so far as to say that "the hand is, together with the word, the essential distinction of man"(4, 198). The junction between hand and textual input device is a vulnerable junction, a space where issues of identity, dependency and control converge. Good, human centered interface design demands that this junction be a protected space, a space that is non-threatening, non-invasive, and encouraging of human expression.

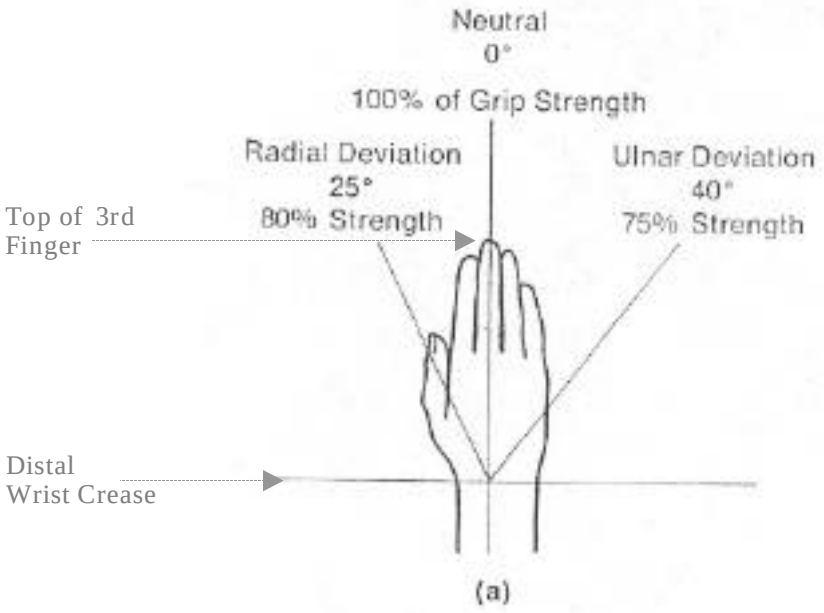
Ergonomic Considerations for Keyboard Design

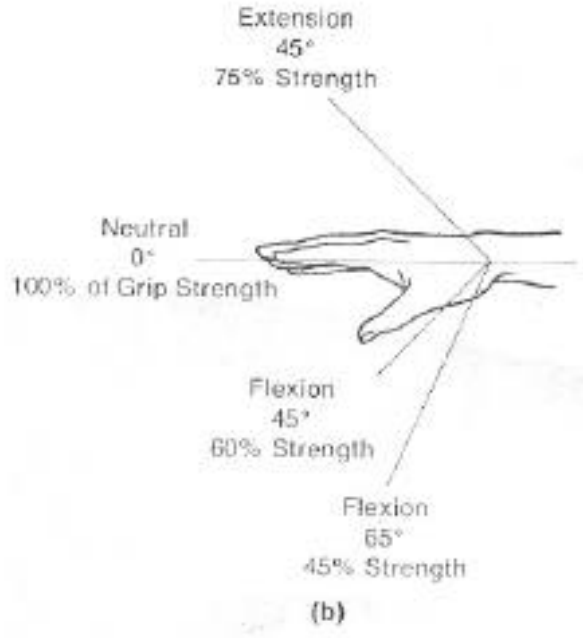
Ergonomics is an interdisciplinary field involving bio-mechanical, physiological, psychological and behavioral considerations. The field's complexity together with human individuality makes pronouncing a design "ergonomically correct" or "ergonomically incorrect" a somewhat pseudoscientific processes. Nevertheless, there exist rules of thumb which can be used as approximate guidelines for "good" ergonomics.

The guidelines presented below are based on the recommendations of Liberty Mutual Group (America's largest workers' compensation ensurer), The National Institute for Occupational Safety and Health (NIOSH), Lisa Tiraboschi (the associate director for Environmental Health and Safety at Dartmouth College), the Eastman Kodak Company Ergonomics Group Health and Environmental Laboratory, occupational therapist Denise Finch, and numerous individuals coping with keyboard related RSIs(29 33 & 31). The primary ergonomic considerations for keyboard design are:

- **Neutral wrist alignment** (See the following page for a visual definition of neutral alignment).
- **Use does not require static posture.** Static posture inhibits the circulation of blood.
- **Use does not constrain the upper body's range of motion.** The "best" posture is person-specific. The more a typing device constrains a person's range of motion, the less chance there is of a healthy posture being achieved.
- **Key actuation does not require excessive or awkward motions and forces.**
- **Typing does not cause pain.**

Neutral Wrist Alignment



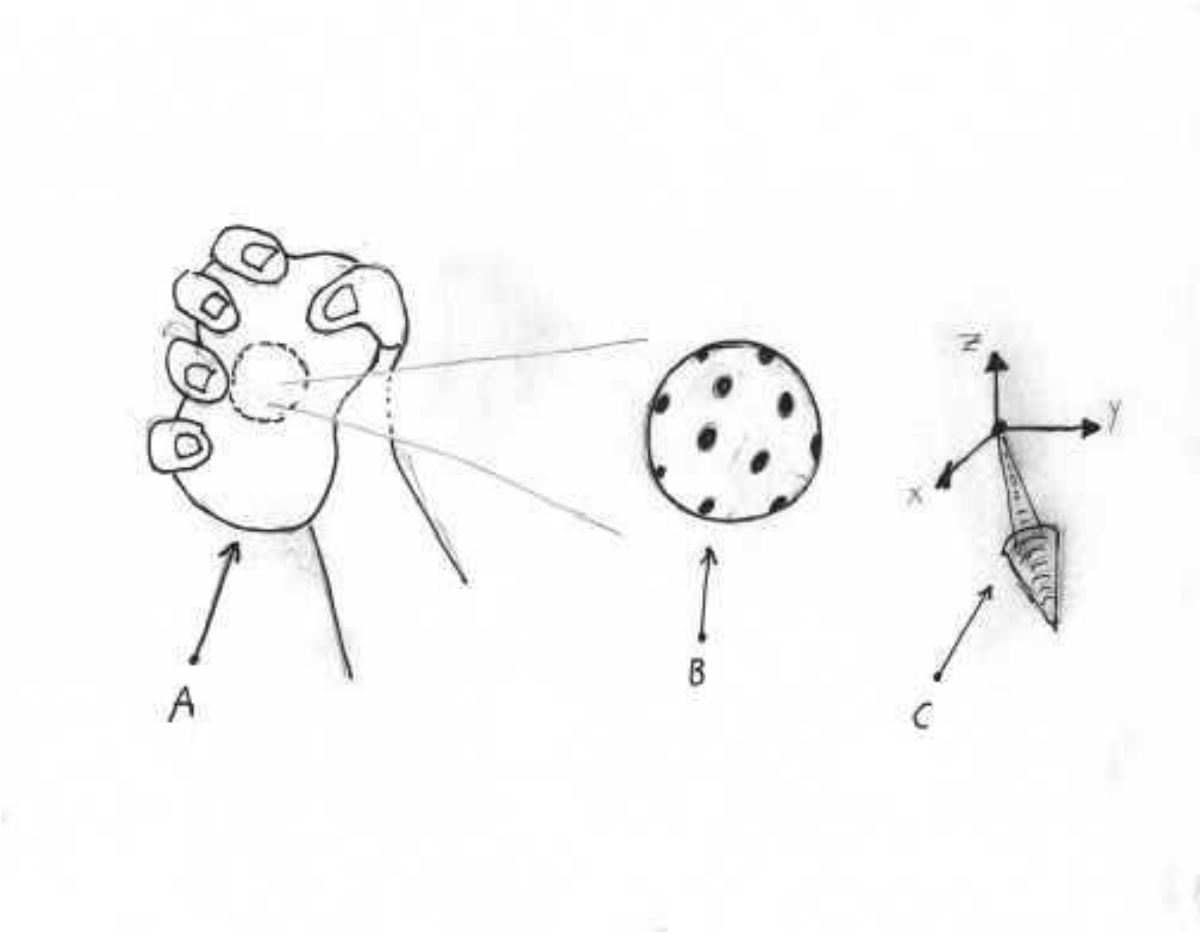


Diagrams reproduced from
"Ergonomics for People at Work"(31, 470)

Initial Brainstorm

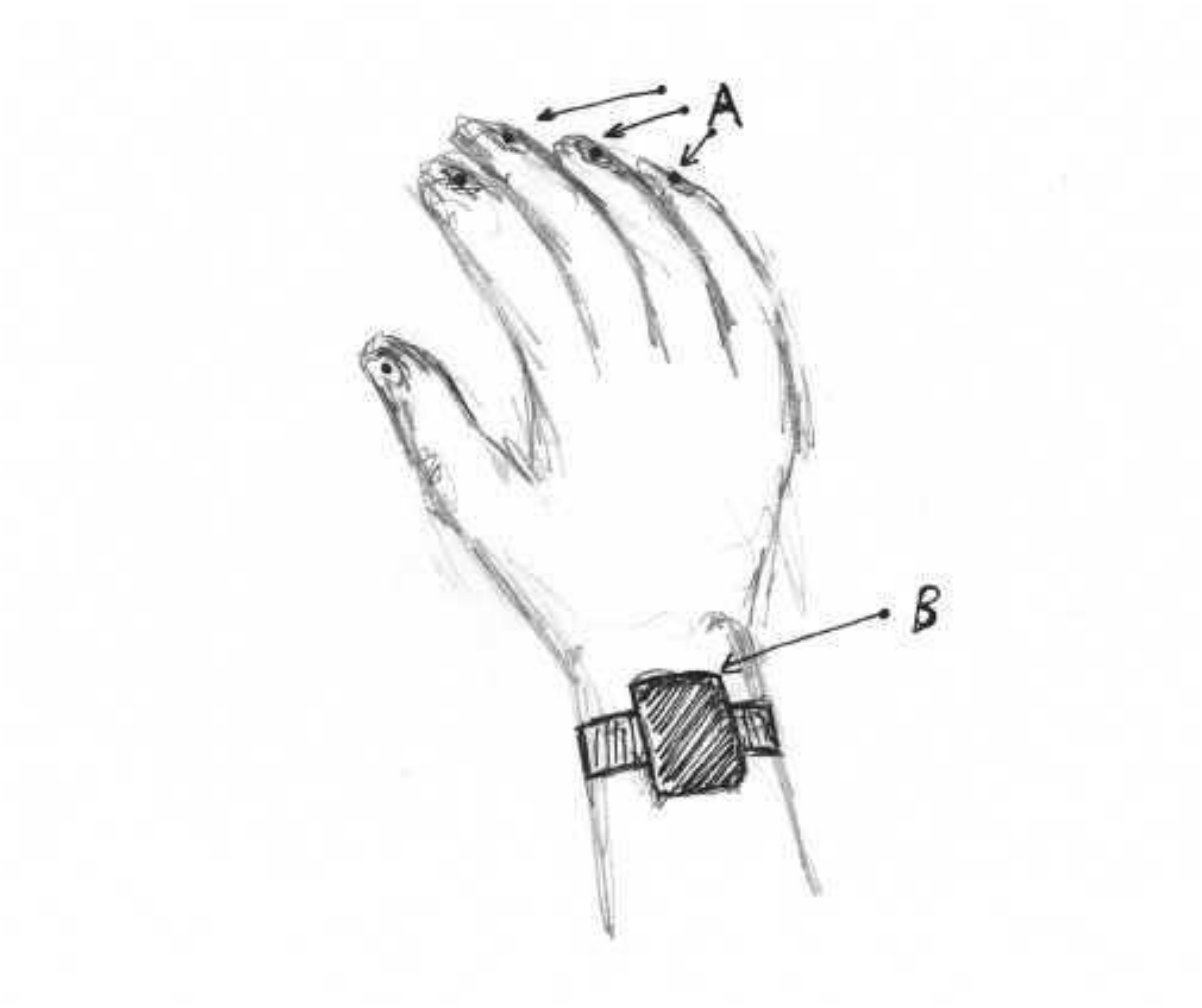
Shown over the next few pages are concept sketches illustrating some of the potential solutions that emerged from initial brainstorming.

"The Adaptive Lump"



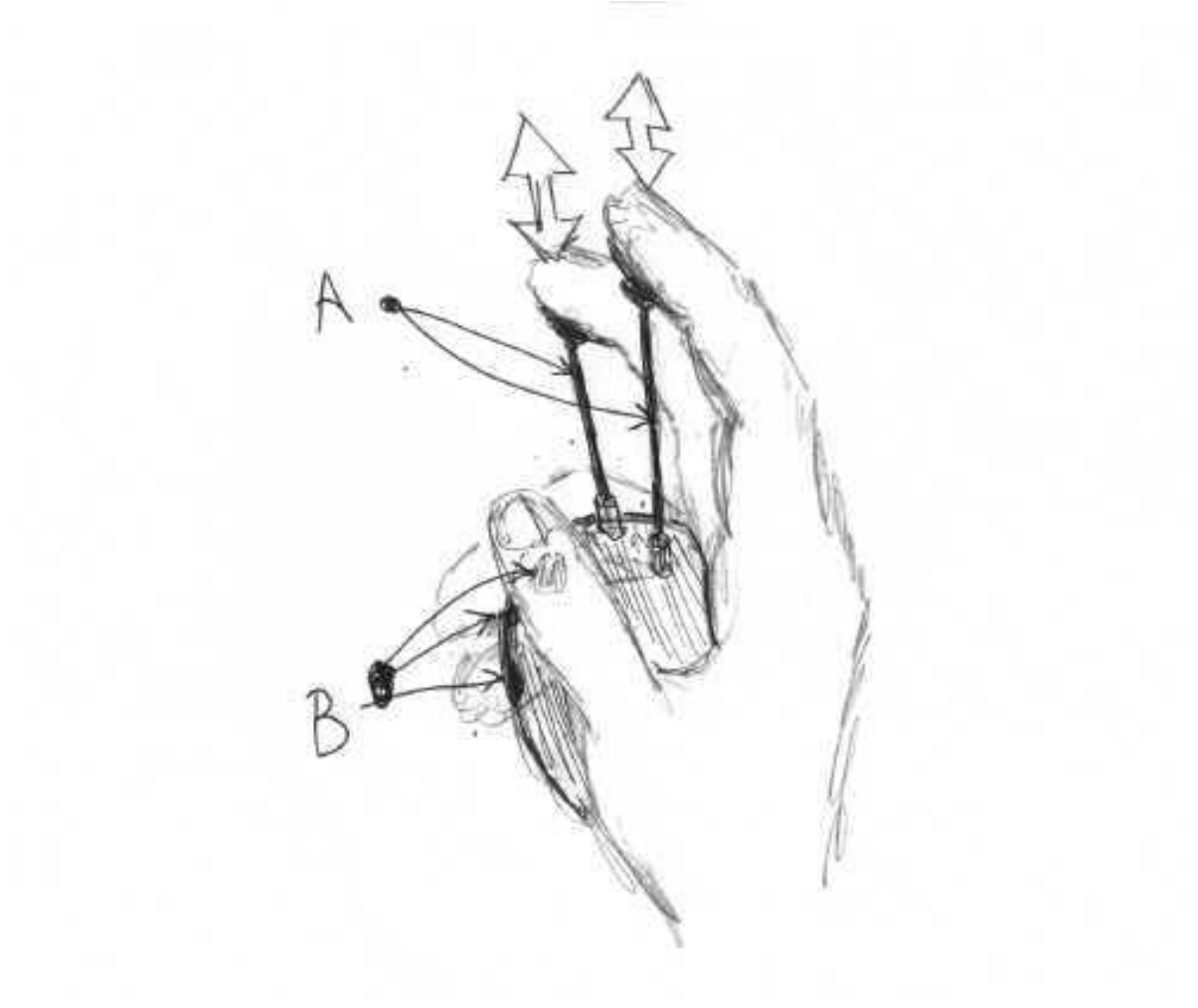
The "Adaptive Lump" has an electronic core covered with pressure sensors. Users mold their own plastic shape around this core, and develop their own finger movement control language. The core uses information from all the pressure sensors to create a 3D-pressure vector that gets mapped to characters by a remote base station.

The "E.M. Field Watch"



The "E.M. Field Watch" is a device (B) worn on the wrist that keeps track of the position of fingers by changes in the field-lines of an electromagnetic field generated around the hand. Hand movements are then mapped to characters. The device might be accompanied by finger rings or units placed on the hand's finger nails (A) to exaggerate the way hand movement distorts the surrounding electromagnetic field.

The "FingerPump"

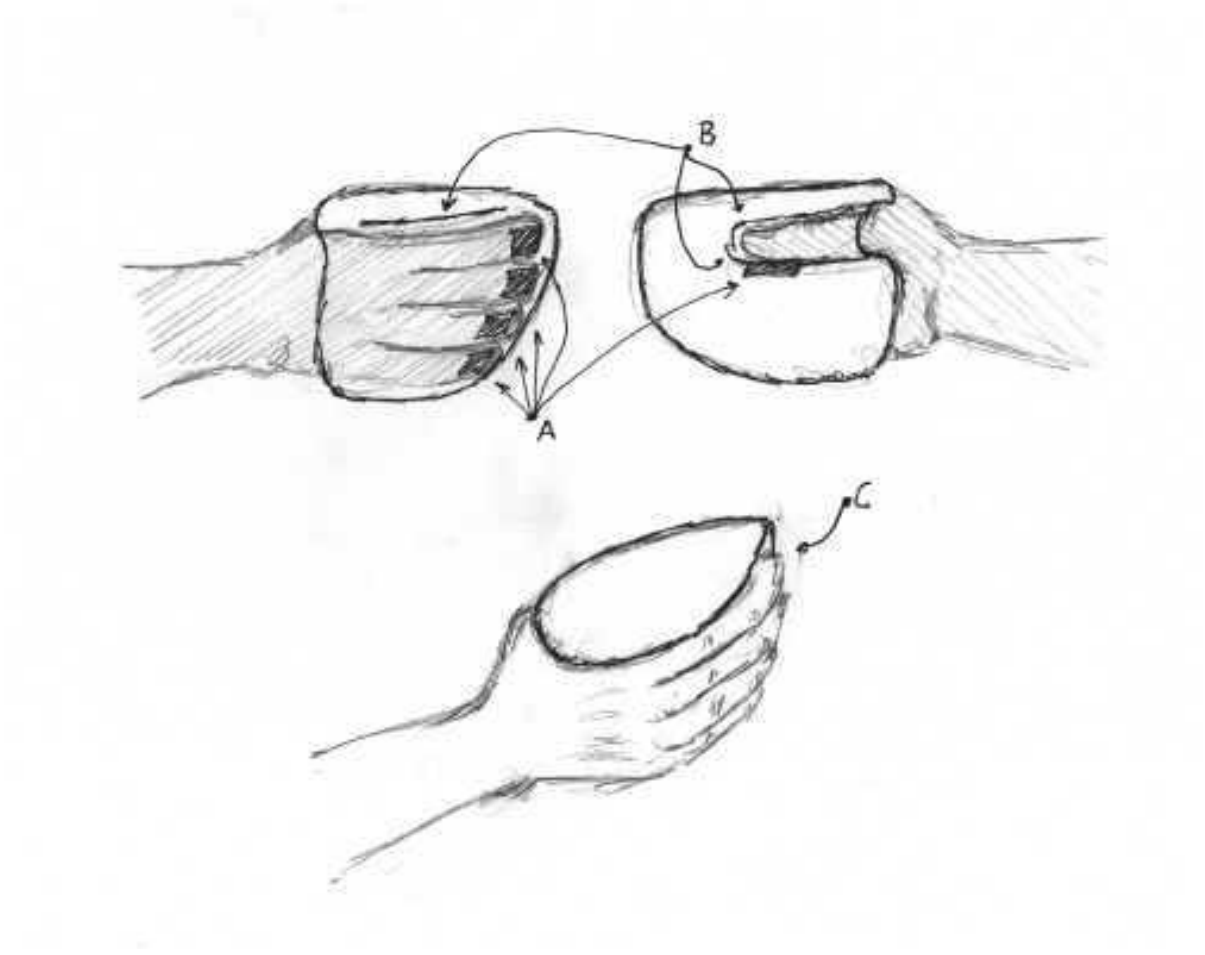


The "Finger Pump" is held by the thumb and the two smallest fingers. Two slider buttons (A) operated by the two larger fingers provide a variety of possible combinations. These combinations--together with push-buttons (B)--create unique combinations corresponding to characters.

The "Data Flute"

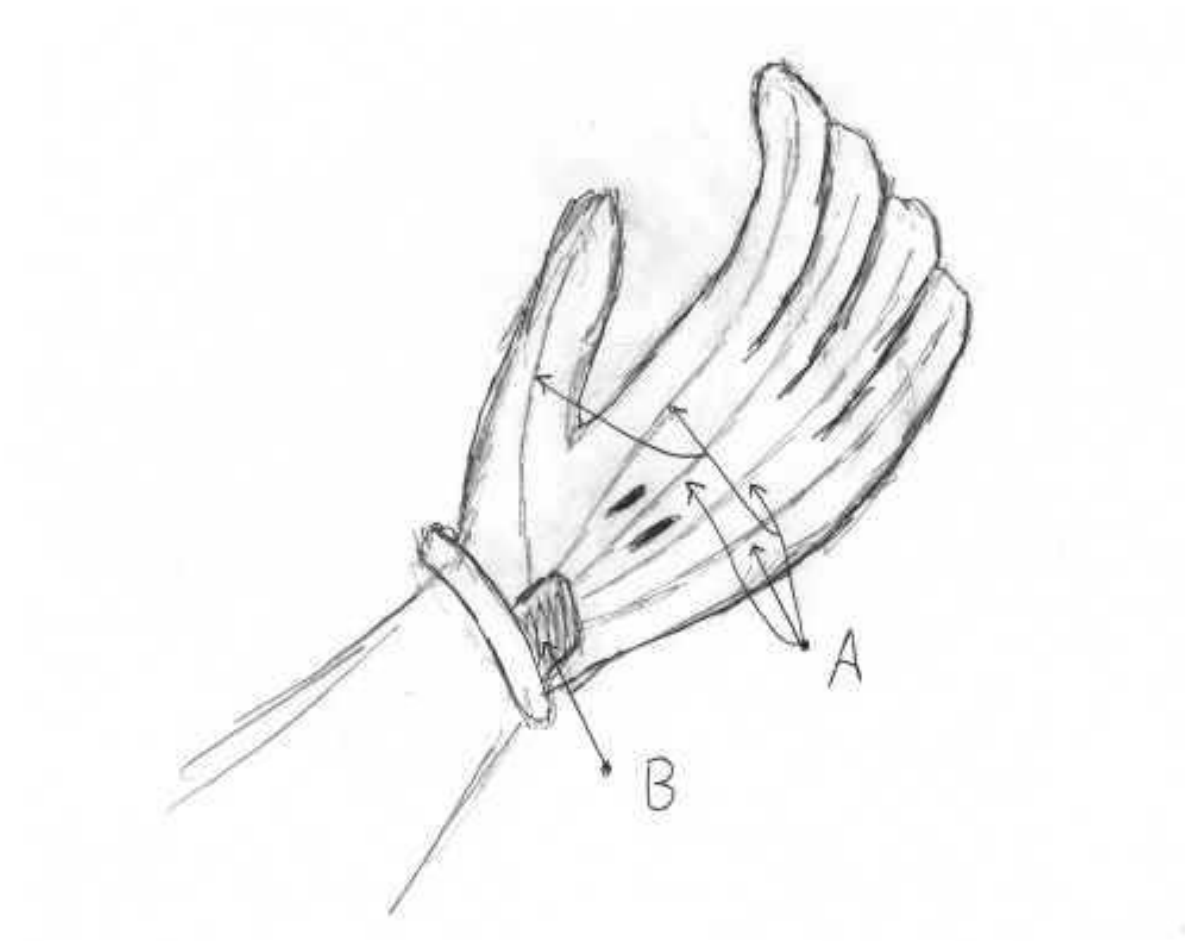
The "Data Flute" is a two handed device with an interface like that of a penny whistle. (Think of how fast the penny whistle can be played!) It is roughly the same size and shape as an expensive pen, and could be stored in a pocket easily.

The "Fruit"



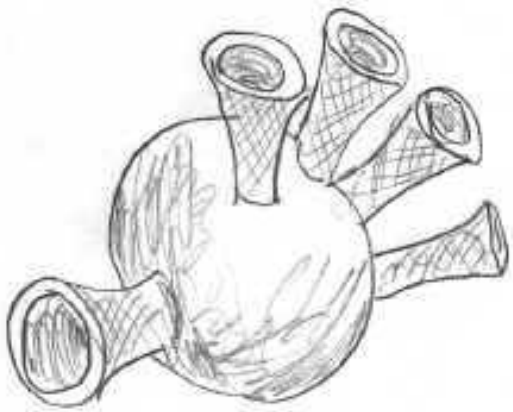
The "Fruit" is designed to fit the hand's resting position, and thus does not require much effort to hold onto. A flange encircling the devices' top helps support the weight of the device (B & C). Each digit is responsible for one or two buttons (A).

The "Mickey Glove"



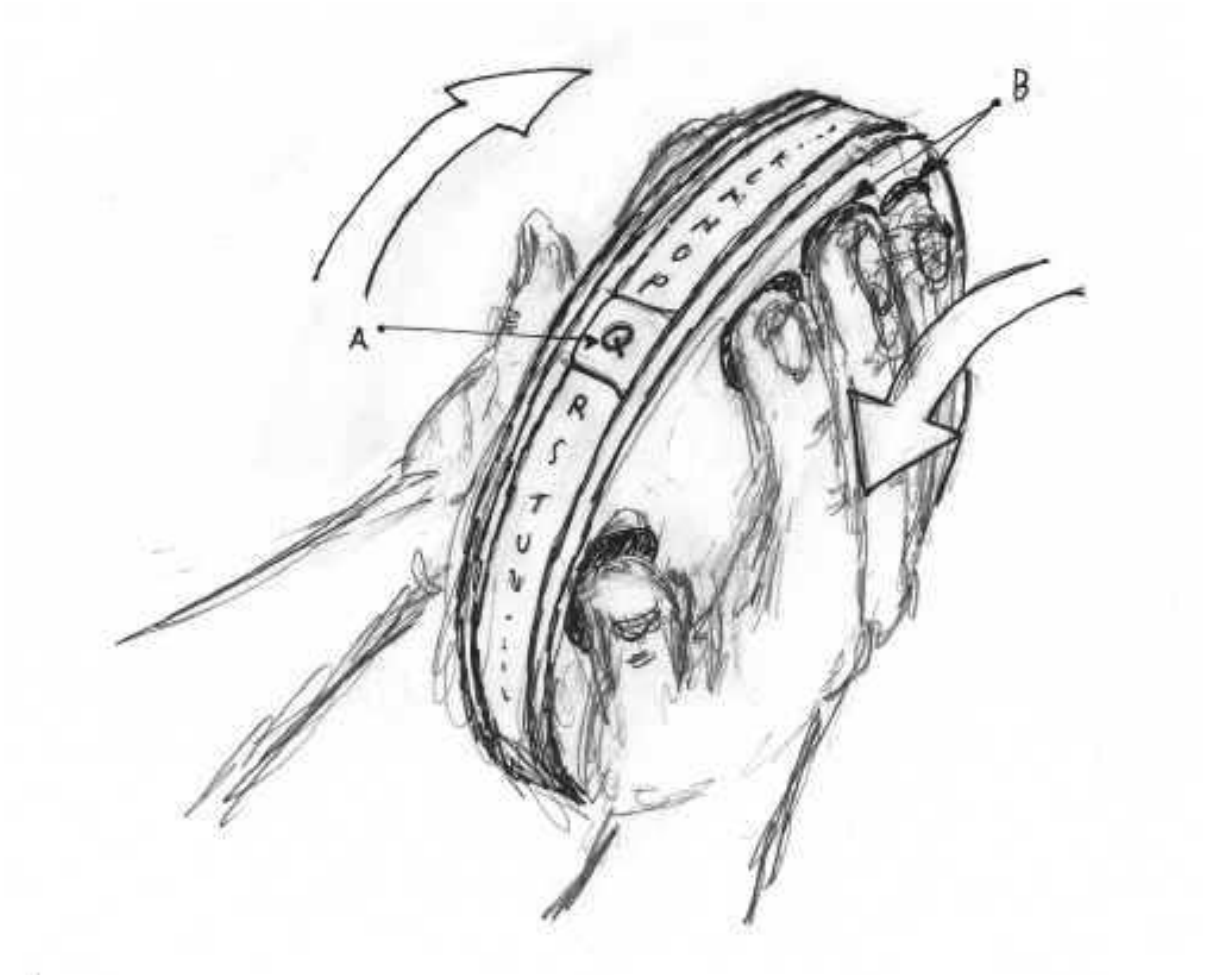
The "Mickey Glove" is a glove similar to the Nintendo Power Glove, but designed to translate finger and thumb positions into characters.

The "Heart"



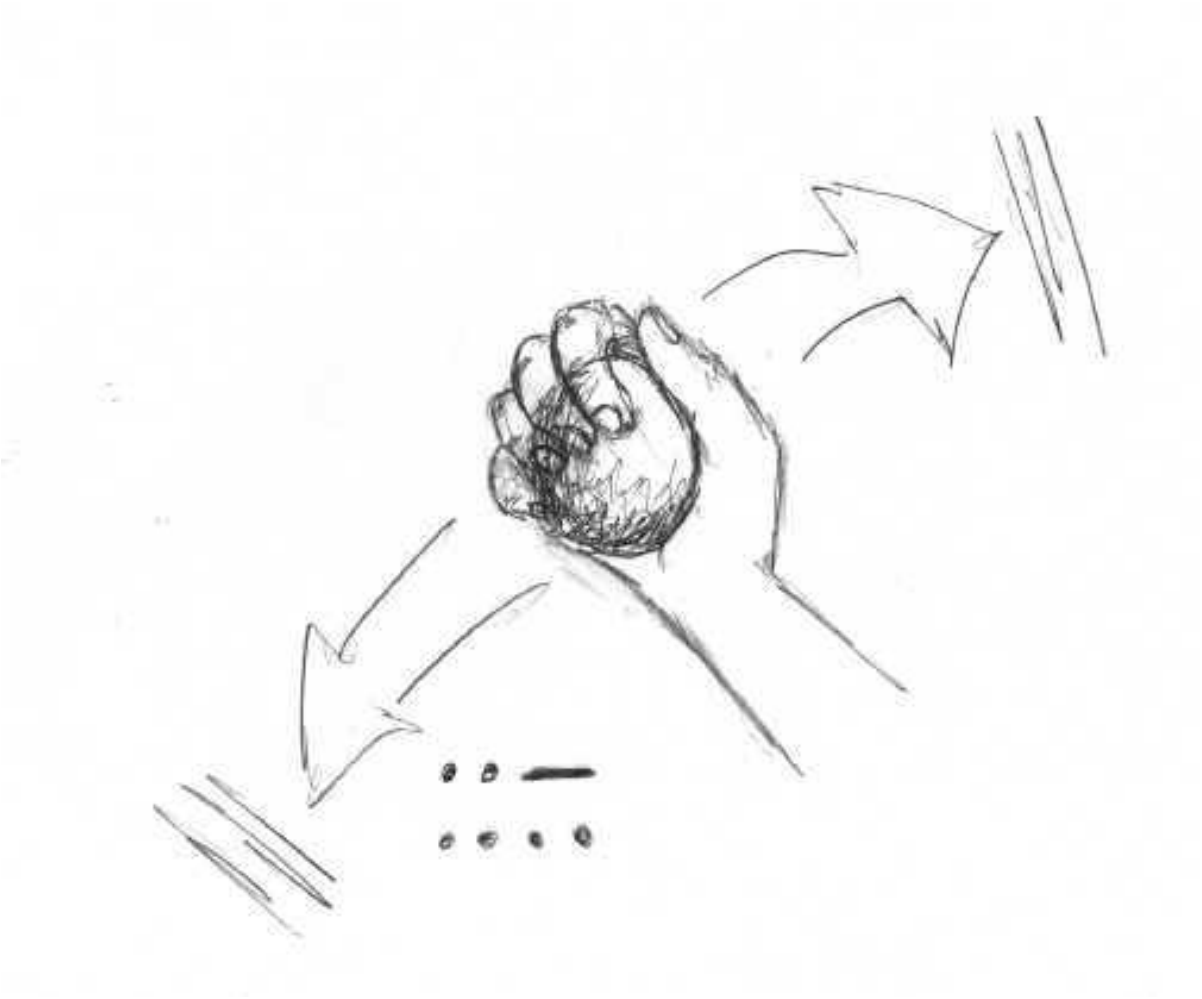
The concept of the "Heart" is that each digit rests inside a socket similar to a Chinese finger trap. Each digit has push/pull control--this would make possible a greater number of finger position combinations than would be possible with, say, one push-button per digit. Releasing the device might prove difficult.

The "Pancake"



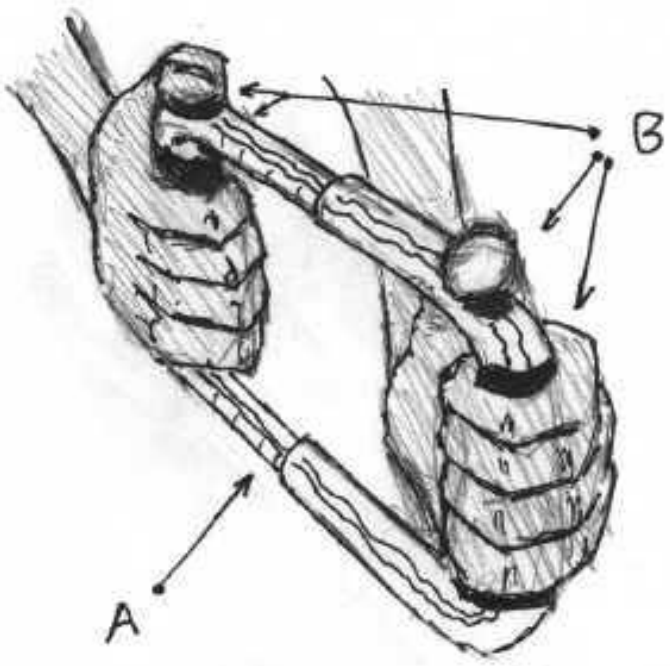
The "Pancake" is a device that uses both mechanical rotation and button actuation to select characters, which appear along the device's perimeter. The character to be selected could appear under a visible crosshairs mark--this would improve the device's learnability and usability. In principle, operation of this device would be like the operation of a tubular slide rule.

The "Salt Shaker"



The "Salt Shaker" houses an accelerometer. By shaking it in a serial pattern such as Morse code, a user could send characters.

The "Saw"



The "Saw" interface crosses the interfaces of trombone and accordion. Sliding hands away from each other (A) would provide one form of control, while buttons on each handle (B) would provide another form of control. The combination of these control types might prove interesting.

The "Strap-O-Type"

The "Strap-O-Type" typing system could involve some combination of pulling shoulder straps and operating actuators placed on these straps. This system could be mounted on suspenders or the straps of backpacks. The system might be useful for interfacing large and heavy computers required to be portable. (Such device could be housed in a backpack).

Typing Habits Survey

Human Interface Design Project Survey Krispin Leydon Jan 23, 2000

Background Information/Benchmarks

Name:_____
What is your profession?

What is your age?

Please trace the outline of your dominant hand on the back of this piece of paper.

Do you have a dominant hand? If so, which one? [L or R]

How would you describe your learning style? Kinesthetic (learning by doing), visual (learning by watching), auditory (learning by listening) or other _____ ?

How fast can you type? [words per minute]

On average how many hours a day do you spend typing? [(<.5) (.5-1) (1-2) (2-3) (<3)]

Do you use a computer for longer periods of time now than you did five years ago?

What sort of device do you use to type?

Do you touch-type, or hunt-and-peck?

Is the keyboard you use to type comfortable? [1 2 3 4 5]

What do you like about it?

What do you dislike about the keyboard you use?

Can you envision scenarios where you might want to send text information, scenarios which are impossible given a standard keyboard interface? If so, please describe these scenarios.

How valuable would the ability to type comfortably while "on the move" be for you?
[1 2 3 4 5]

Does the notion of a light-weight electronic device attatched to your body or sewn into your clothing bother you? If so, what about these possibilites bothers you?

HID Project	Kristin Leydon															
TYPING HABITS SURVEY RESULTS																
Name	Profession	Age	Dominant Hand	Learning Style	Hours	TypingDaily	TypeMor	ThanSyearsAgo	TouchType?		WPM	Typing Device	Comfortable?	Pros?	Cons?	NewTypingScenarios	Type&MoveValue	DoinsAttachment	BotherYou?
Brian Fife	ES Student	21			2to3		Y		Y		95	pc kbd	4	keyResponse	laptop kbd	limited space or on move		no.	
Michael Fromberger	Software Engineer	26	R	K,V,A	>3	N	Y		Y		120	mac kbd	4	keyResponse, full size, uniform pressure	asymetric hand use	note-taking while upright and moving, one handed operation		3	sewn-yes, strap-no
Lisa Trabocchi	SHS Asst. Director	44	R	A	2to3		Y		Y			mac kbd	5	Iris hands well	stupid keypad	none envisioned		3	sewn-yes, wise maybe
Sharona Davis	ES/CS Student	21	R	K	2to3		Y				65	kbd	3	parts job done	hurts wrists, not comfy	none envisioned		5	yes-invasive, Too personal.
Jeannie Lee	Composer, Student	23	R	K	.5to1		Y		Y		70	kbd	3	small size	too straight, no wrist support	memos, spontaneous ideas		3.8	yes-fear of breakage
Charlie Sullivan	ES Prof	35	R		>3		N				75	ergo kbd	4	light touch	keys too torn out abruptly			4	yes-want to get away from technology sometimes. Nerdy image.
Ashok R	ES Grad Student	24	R	K	1to2		Y		Y		65	ergo kbd	3	keys go deep, quiet, broad. Not like i-mac	hurts wrists	spontaneous ideas		5	looks geeky
Jerry Rutter	Teacher, Archaeologist	53	R	V,Reading	1to2		Y		N		65	mac kbd	2	not much	too small, hit two keys at once	Filling out database format, replying to simple questions, onsite note taking		4	no.
GeneCesari	Retired Head of School	69	R	K,V,A	.5to1		Y		N		25	ergoprosex kbd	3	legibility, size	nothing comes to mind	taking notes, reminders, recording ideas when not near a keyboard		3	sewn-no, strap-yes. Awkward, uncomfy, worry about damage
Sumit Borah	Student, LabTechnician	23	R	K	1to2		Y		Y			mac kbd	4	ergonomic, curved, neutral hand position	no flexible keymap	nothing comes to mind		1	no.
Suneth J	Student	23	R	K,V	2to3		Y		Y		75	chars mac kbd	4	soft auditory feedback	keys slow to return, can't type w/o wrist rest			3	only if interferes with fashion
Lauren Newton	Perpetual student	23	R	V	1to2		N		Y			kbd	4	quiet	trackpad in way	eating and typing		2	ok attatched to clothes, not sewn in.
Michelle Ort	student	23	R	K	1to2		Y		Y		80	kbd	3	I know it	probably not good for hands wrists cause not new ergo models	???		2	no.
Mike Brennan	student	21	R	V,K	1to2		Y		N		50	pc kbd	3	soft keys, get response	worried about carpel tunnel syndrome	for typing equations, derivations		5	yes--want time away from technology. Aprehensive. Like cell phones
Rupa Mukherjee	Health Care Analyst	23	R	Mostly	K,V,A	>3	Y		Y		70	kbd	3	elevated, good position	key jam, can't find a "right" position on the desk,need flat stable surface	typing and drinking hot chocolate		4.5	sewn-yes,if bulky
neena shin	student	21	R	K	>3		Y		Both		35-60	mac kbd		keys press easily	no page-up/down	no, not yet.		2	no.
alison jeffe	student	20	R	V	>3		Y		Y		65	kbd	5	good keys	nothing wrong	no		1	no.
Sam Feakins	student	20	hope	V,Reading	>3		Y		Y		fast	kbd	3	good keys--don't like wierd ergo boards	little small-wrists hurt	no		5	no.
John Cruz	Marketing Consultant	23	R	K,V	>3		Y		Y		120	dat	5	keys don't stick	loud	no		1	Yes -- worried about size and weight
Colin Sandberg	Engineer	23	R	writing	1to2		Y		Y		15-25	pc kbd	3	good keys	wrists bent unnaturally	while teleconfrencing		3	no.
Sarah Kellmenson	Med Student	23	R	V,K	.5to1		N				medium	laptop mac kbd						3	sewn in yes, attatched to hands, no.
Jason Erk	Elementary Teacher	23	R	K	1to2		Y		Y		75	kbd	1	familiar	get tired	typing on workout bike, journaling while traveling, recording thoughts while interview		5	no.

Shape Survey

Human Interface Design Project Survey Krispin Leydon Jan 23, 2000

Initial Shape Survey Name:_____

1. Individual Shape Critiques

How comfortable is it to hold? [1 2 3 4 5]
1) 2) 3) 4) 5) 6) 7) 8) 9) 10) 11) 12)

How hard was it to pick up the shape and have it set in a comfortable position? [1 2 3 4 5]?
1) 2) 3) 4) 5) 6) 7) 8) 9) 10) 11) 12)

How hard was it to put down? [1 2 3 4 5]
1) 2) 3) 4) 5) 6) 7) 8) 9) 10) 11) 12)

What do you like about this shape; what about it "works"?
1) 2) 3) 4) 5) 6)
7) 8) 9) 10) 11) 12)

What bothers you about this shape; how does it not work?
1) 2) 3) 4) 5) 6)
7) 8) 9) 10) 11) 12)

Can you imagine typing with this shape? [Y N]
1) 2) 3) 4) 5) 6)
7) 8) 9) 10) 11) 12)

2. Overall Shape Critiques

Which is your favorite shape?

Which shape felt most comfortabe?

Which shape can you most easily envision yourself typing with?

What would you call this device?

What weight feels right for the device? [#of quarters]

If you could use two of these devices--one per hand, and tye 17% faster, would you choose the two-handed system over the one-handed system?

If you had a choice between a cord-tethered device that could be used indefinitely, or a wireless device with a battery that had to be changed once a month, which would you prefer?

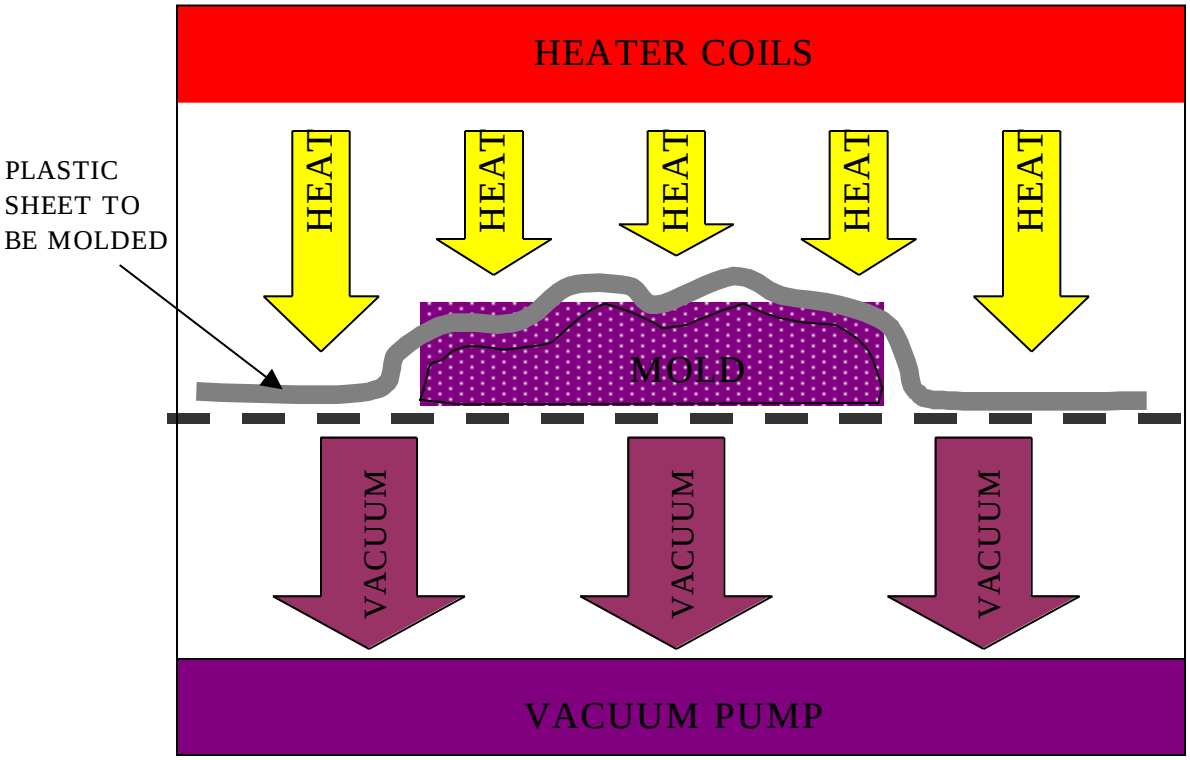
Do you have any other comments/suggestions/ideas?

HID Project	Xriapin Leydon											
INITIAL SHAPE SURVEY RESULTS		8										
Individual Shapes:												
Shape	Comfortability	Picking Up	Putting Down	What do you Like? What "works"?	What don't you like? What doesn't "work"?	Can you imagine typing with this shape?						
1	3.9	2		1.5	soothing, simple, fits,comfy	impractical,toosmall,hold&press-,unused area	3N, 5Y					
2	4.2	2.5		2.3	thumb+,stays put, fingers+, secure	thumb-,top annoying,	8Y					
3	4	2.1		1.5	atural feel,fingers+, thumbmove++	talltoo, hold slips, Restrictsindexf,fingers-,hold,nothing,morespaceneeded F2&T	2N,6Y					
4	2	2.9		1.8	simple,fingers+,nothing,simple,thumb+,	boxy,orient-, unsteady,comfy-,slips,thumb-,wastespace, butt too thick	7N, 1Y					
5	3	2.5		1.6	simple ,nothing,palm+,comfort,fingers+	bulky,too big, slips, thumb-,hold-	3N, 5Y					
6	2.75	2.8		1.75	thumb+ , nothing,contour, fingers++,	hold-,large,toofree,clumsy	6N, 2Y					
7	3	2.8		1.9	ingers+,hold+,compact,palm,simple	toosmall,nobuttonspace,unstable, thumb&finger edges sharp,thumb-,	4N, 4Y					
8	2.8	3		1.9	compact,nothing,finger+,comfy	toosmall, unstable, thumb-, 2small, hold-, palm-, no guides for fingers	6N, 2Y					
9	2.7	3.2		1.7	ymetric,nothing,wide,finger+,balanced	pancake shape, too big, fingers-, hold-, grip-, slips	4N, 2Y					
10	3.5	2.4		2	symetric,nothing, like videogame control, options+	videogame-like, two handed, toothick, not appropriate, wrist-, confining, shoulders, angles wrong	3N, 5Y					
11	3	2.9		2.1	ike video game, nothing, wide, nothing, small	videogame-like, flat, hold-, thin, wrists bent, confining, two handed	2N,6Y					
12	2.8	2.1		2.25	legant, simple, familiar, symetric,size+, interesting, portable	nintuitive, narrow, too weird, rolls, socially unacceptable, how to hold up,	2N,6Y					
Overall Shape Critiques:												
Name	Favorite Shape	MostComfortable	Device Name?	Easiest to Imagine Typing With	Tethered or Wireless?	Acceptable weight for the device?	2or1 handed?	Comments/Suggestions/Ideas?				
Brian Fife		1	12, 3, 12, 11, 7				43 Oz = 258 g					
Michael Frombergei	2, 12	2, 12	DataWhistle, Keygrip	12	Wireless		20 Oz = 120 g	1				
Lisa Traboschi	2, 1	2	KeyType	2	Wireless		14 Oz = 84 g	2				
Shanna Davis	3	3	trgthand	3 or 2	Wireless		34 Oz = 204 g	1	colors!			
Jeannie Lee	2, 1, 7	1	The DigeryBoard	1	with strap	Wireless, both would be nice.	8 Oz = 48 g	1, mirror image				
Charlie Sullivan	2!!!	3	handkey	2	wireless		16 Oz = 96 g	1				
Ashok R	3	3	Palmer	3			33 Oz = 198 g	1	extensions! PDA!			
Jeremy Rutter	5, 12	2,3	TeleDactyl (greek Tele(distance) daktylos(finger)	5	wireless		25 Oz = 150 g	1				

Thermaforming

"Thermaforming" is a process by which plastic shapes of relatively thin uniform thickness can be molded quickly, easily and cheaply. During thermaforming, a sheet of Acrylic-PVC plastic--such as Kydex®--is simultaneously heated from above and subjected to a vacuum pressure from below. Heat causes the plastic to become temporarily pliable and this makes it possible for vacuum pressure to "suck" the plastic sheet over a three dimensional mold, as is demonstrated in the picture below:

The Thermaformer in Action



FIRMWARE -- Prototype 1

The following programs were written/adapted for driving the 8052 microprocessor and accessing a serial port via RS-232 communication:

key.c

```
/*
KEY.C
Krispin Leydon
Jan 2000


FUNCTIONAL DESCRIPTION:
This program is the controlling software for a one handed 7 key "chording"
keyboard.
*/



/*Include Directives*/
#include <reg51.h>                              /* define 8051 registers */
#include <stdio.h>                              /* define I/O functions */



/*Defines*/
#define bool bit
#define byte char



/*Static Variable & Constants*/
static const byte TL0_VAL = 0x66; /*TL0 & TH0: -922 base10*/
                          /*(to count .001sec)*/
static const byte TH0_VAL = 0xFC;
static const byte TH1_VAL = 0xFD;  /*9600 Baud (with 11.0592MHz Xtal)*/



/*Global Variables & Constants*/
data byte prev_chord = 0;
data byte current_chord = 0;
data unsigned byte settle_time = 150;
data unsigned short int time_before_repeat = 500;
data unsigned short int repeat_delay_time = 100;
bdata bit time_up = 0;
idata char chord_map[129];



/*Prototypes*/
```

```
bool chord_is_changing (); /*Returns 1 when a new chord begins/ends*/
bool settled (); /*Returns 1 when a chord is maintained for x msec*/
void auto_repeat (); /*Wait to see if "typematic" action is necessary*/
char translate (byte chord); /*Maps chords to chars*/
void send_char (); /*Sends char to wherever*/
void init_chord_map (); /*Creates a chord map*/



/*TIMER0_ISR -- Interrupt service routine*/
/*Fires each millisecond, then sets flag "time_up" to 1*/
/*timer is turned on and flag reset from within other functions.*/
void timer0_isr (void) interrupt 1
      {
      /*Set flag*/
      time_up = 1;

      /*Turn timer 0 off*/
      TR0 = 0;

      /*Reload Timer0 Value*/
      TL0 = TL0_VAL;
      TH0 = TH0_VAL;

      return;
      }



/*MAIN*/
void main ()
      {
      /*Initialize 8051 serial I/O stuff*/
      TMOD = 0x21;
      TL0 = TL0_VAL;
      TH0 = TH0_VAL;
      SCON = 0x5a;
      TCON = 0xc0;
      TH1 = TH1_VAL;
      PT0 = 0;   /*Set timer 0 interrupt to low priority*/
      ET0 = 1;   /*Enable timer 0 interrupt*/
      EA = 1;    /*Global interrupt enable*/


      /*Initialize Other 8051 stuff*/
      P1 = 0xFF;


      /*Map chord combos to characters*/
      init_chord_map();


      /*Initial Prompt*/
        printf("\nType!\n") ;


      /*main (infinite) loop*/
```

```
      while(1)
            {
            /*Write All 1s to Port1 Input Bits -- if any are 0 when read,
means input happened*/
            P1 = 0xFF;

            /*Test to see if user has begun to enter a new chord*/
            if (chord_is_changing())
                  {
                        /*New Chord Started:*/
                        /*Wait until the value of P1 settles.*/
                  /* (Wait until the value of P1 doesn't
                        /* change for a period of x msec).*/

                        while( !settled() )
                              {
                              }

                  /*Send character corresponding to chord to*/
                  /*wherever.  If chord is held for a period*/
                  /*of y msec, start repeating the chord (typmatic
                  action--yeah!).*/
                        if (current_chord != 0x00)
                              {
                        send_char();
                        auto_repeat();
                        }
                  }
            }
      }



/*CHORD_IS_CHANGING*/
bool chord_is_changing ()
      {
      if ( current_chord != ((P1 ^ 0xFF) ^ 0x00) )
            {
            prev_chord = current_chord;
            current_chord =  ((P1 ^ 0xFF) ^ 0x00);
            return (1);
            }

        else
            return (0);
      }









/*SETTLED*/
```

```
bool settled ()
      {
      data unsigned byte i;

      for ( i = 1; i < settle_time; i++)
            {
            time_up = 0;
            TR0 = 1;   /*Start timer 0*/
            while (!time_up)
                  {
                  if (chord_is_changing ())
                        {
                              /*Chord hasn't settled.*/

                        /*Turn timer 0 off*/
                        TR0 = 0;

                        /*Reload Timer0 Value*/
                              TL0 = TL0_VAL;
                        TH0 = TH0_VAL;

                        return 0;
                        }
                  }
            }
      /*Chord has settled!*/
      return 1;
      }



/*REPEAT*/
void auto_repeat ()
      {
      data unsigned short int i;

      /*Wait for chord to be held for a sufficient ammount of time*/
      /*Return if chord changes during this wait period*/
      for ( i = 1; i < time_before_repeat; i++)
            {

            time_up = 0;
            TR0 = 1;   /*Start timer 0*/

            while (!time_up)
                  {
                  if (current_chord != ((P1 ^ 0xFF) ^ 0x00))
                        {
                              /*Turn timer 0 off*/
                        TR0 = 0;

                        /*Reload Timer0 Value*/
                              TL0 = TL0_VAL;
                        TH0 = TH0_VAL;

                        return;
                        }
```

```
                  }
                }

      /*Chord has been held a while; auto-repeat character*/
      while ( current_chord == ((P1 ^ 0xFF) ^ 0x00) && current_chord !=
0x00 )
            {
            send_char();

            /*Delay after each autorepeat*/
            for ( i = 1; i < repeat_delay_time; i++)
                  {

                  time_up = 0;
                  TR0 = 1;   /*Start timer 0*/

                  while (!time_up)
                        {
                        if (current_chord != ((P1 ^ 0xFF) ^ 0x00))
                                    {
                                          /*Turn timer 0 off*/
                                    TR0 = 0;

                                    /*Reload Timer0 Value*/
                                          TL0 = TL0_VAL;
                                    TH0 = TH0_VAL;

                                    return;
                                    }
                        }
                  }
            }

      return;
      }



/*TRANSLATE*/
char translate (byte chord)
      {
      return(chord_map[chord]);
      }



/*SEND_CHAR*/
void send_char ()
      {
      printf("\n");
      printf("%c", translate(current_chord));
      printf("\n");
      }


/*INIT_CHORD_MAP*/
void init_chord_map ()
```

```
      {
      chord_map[1]='a';
      chord_map[2]='b';
      chord_map[3]='c';
      chord_map[4]='d';
      chord_map[5]='e';
      chord_map[6]='f';
      chord_map[7]='g';
      chord_map[8]='h';
      chord_map[9]='i';
      chord_map[10]='j';
      chord_map[11]='k';
      chord_map[12]='l';
      chord_map[13]='m';
      chord_map[14]='n';
      chord_map[15]='o';
      chord_map[16]='p';
      chord_map[17]='q';
      chord_map[18]='r';
      chord_map[19]='s';
      chord_map[20]='t';
      chord_map[21]='u';
      chord_map[22]='v';
      chord_map[23]='w';
      chord_map[24]='x';
      chord_map[25]='y';
      chord_map[26]='z';
      chord_map[127]='*';
      }
```

Startup.a51

Written by Keil, minor modifications made for utelizing the additional internal memory of the 8052 chip. (The code was written for the 8051 microprocessor).

```
;------------------------------------------------------------------------
-----
;  This file is part of the C51 Compiler package
;  Copyright (c) 1995-1996 Keil Software, Inc.
;------------------------------------------------------------------------
-----
;  STARTUP.A51:  This code is executed after processor reset.
;
;  To translate this file use A51 with the following invocation:
;
;     A51 STARTUP.A51
;
;  To link the modified STARTUP.OBJ file to your application use the
following
;  BL51 invocation:
;
```

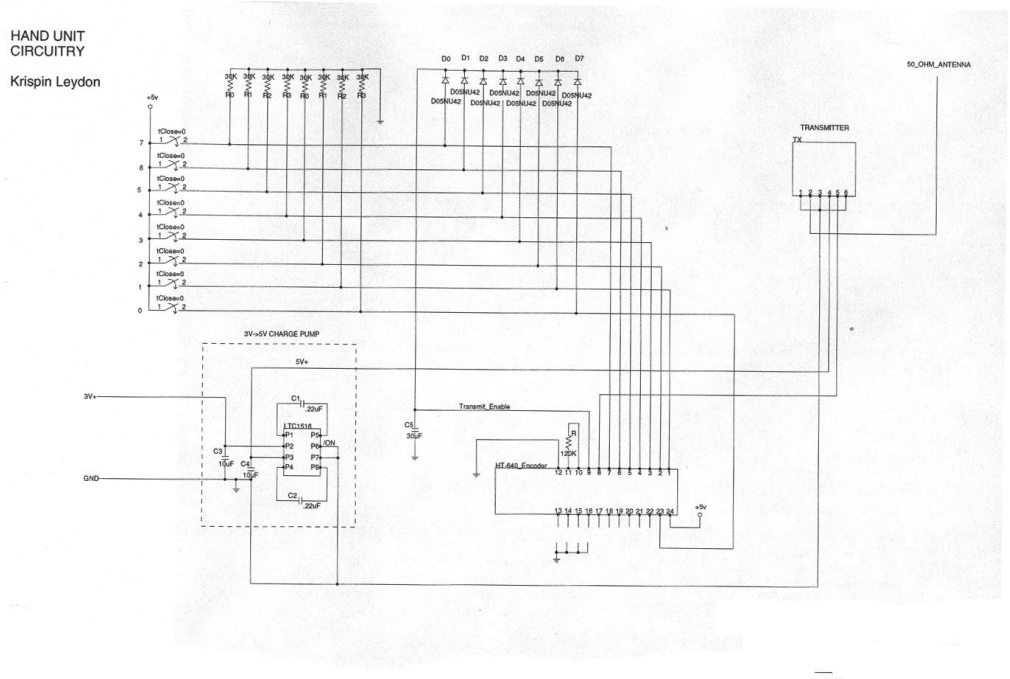
```
;     BL51 <your object file list>, STARTUP.OBJ <controls>
;
;
;  MODIFIED  -
;  slight modification for CSEG AT 0 changed to CSEG AT 8030h
;  to be used for ES 62 board - c. okino 11/10/99
;  (This is as per Keil recommendations for adjusting "origin"
;
;
;------------------------------------------------------------------------
-----
;
;  User-defined Power-On Initialization of Memory
;
;  With the following EQU statements the initialization of memory
;  at processor reset can be defined:
;
;           ; the absolute start-address of IDATA memory is always 0
;IDATALEN   EQU   80H   ; the length of IDATA memory in bytes.
;*******MODIFIED FOR 8052 (There is twice the internal memory*********
IDATALEN    EQU   100H  ; the length of IDATA memory in bytes.

;
XDATASTART  EQU   0H    ; the absolute start-address of XDATA memory
XDATALEN    EQU   0H    ; the length of XDATA memory in bytes.
;
PDATASTART  EQU   0H    ; the absolute start-address of PDATA memory
PDATALEN    EQU   0H    ; the length of PDATA memory in bytes.
;
;  Notes:  The IDATA space overlaps physically the DATA and BIT areas of
the
;          8051 CPU. At minimum the memory space occupied from the C51
;          run-time routines must be set to zero.
;------------------------------------------------------------------------
-----
;
;  Reentrant Stack Initilization
;
;  The following EQU statements define the stack pointer for reentrant
;  functions and initialized it:
;
;  Stack Space for reentrant functions in the SMALL model.
IBPSTACK    EQU   0     ; set to 1 if small reentrant is used.
IBPSTACKTOP EQU   0FFH+1      ; set top of stack to highest location+1.
;
;  Stack Space for reentrant functions in the LARGE model.
XBPSTACK    EQU   0     ; set to 1 if large reentrant is used.
XBPSTACKTOP EQU   0FFFFH+1; set top of stack to highest location+1.
;
;  Stack Space for reentrant functions in the COMPACT model.
PBPSTACK    EQU   0     ; set to 1 if compact reentrant is used.
PBPSTACKTOP EQU   0FFFFH+1; set top of stack to highest location+1.
;
;------------------------------------------------------------------------
-----
;
;  Page Definition for Using the Compact Model with 64 KByte xdata RAM
```

```
;
;  The following EQU statements define the xdata page used for pdata
;  variables. The EQU PPAGE must conform with the PPAGE control used
;  in the linker invocation.
;
PPAGEENABLE EQU   0     ; set to 1 if pdata object are used.
PPAGE       EQU   0     ; define PPAGE number.
;
;------------------------------------------------------------------------
-----

            NAME  ?C_STARTUP


?C_C51STARTUP     SEGMENT   CODE
?STACK            SEGMENT   IDATA

            RSEG  ?STACK
            DS    1

            EXTRN CODE (?C_START)
            PUBLIC      ?C_STARTUP

            CSEG  AT    0h  ;   Modified for ES 62 Board **Modified back!
-KL
?C_STARTUP: LJMP  STARTUP1

            RSEG  ?C_C51STARTUP

STARTUP1:

IF IDATALEN <> 0
            MOV   R0,#IDATALEN - 1
            CLR   A
IDATALOOP:  MOV   @R0,A
            DJNZ  R0,IDATALOOP
ENDIF

IF XDATALEN <> 0
            MOV   DPTR,#XDATASTART
            MOV   R7,#LOW (XDATALEN)
  IF (LOW (XDATALEN)) <> 0
            MOV   R6,#(HIGH XDATALEN) +1
  ELSE
            MOV   R6,#HIGH (XDATALEN)
  ENDIF
            CLR   A
XDATALOOP:  MOVX  @DPTR,A
            INC   DPTR
            DJNZ  R7,XDATALOOP
            DJNZ  R6,XDATALOOP
ENDIF

IF PPAGEENABLE <> 0
            MOV   P2,#PPAGE
ENDIF
```

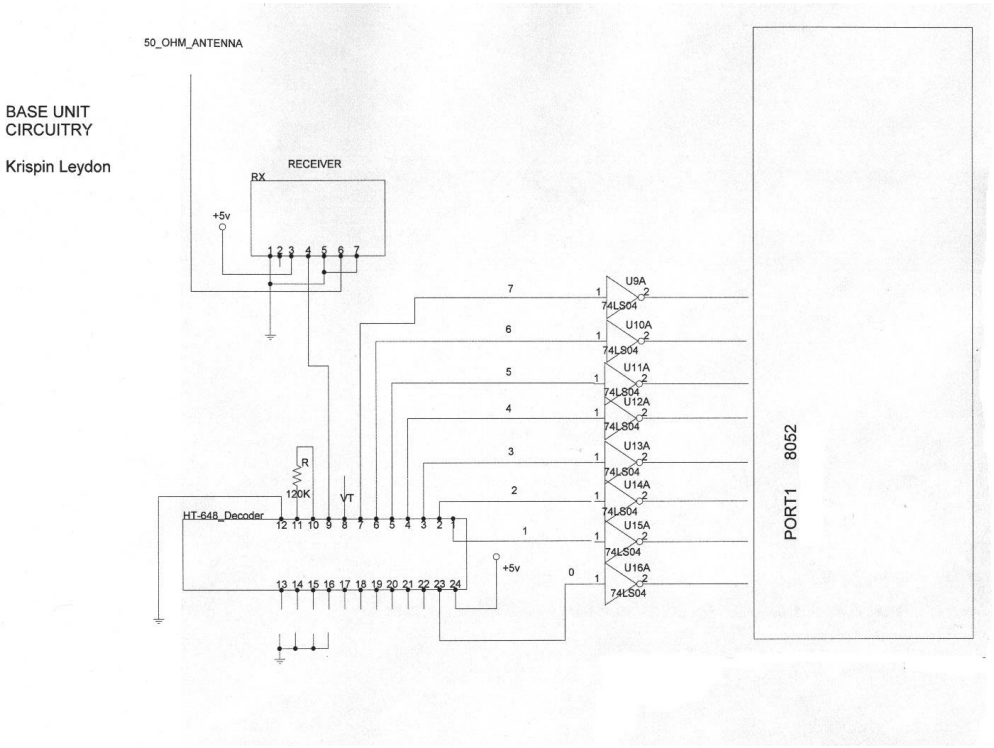
```
IF PDATALEN <> 0
            MOV   R0,#PDATASTART
            MOV   R7,#LOW (PDATALEN)
            CLR   A
PDATALOOP:  MOVX  @R0,A
            INC   R0
            DJNZ  R7,PDATALOOP
ENDIF

IF IBPSTACK <> 0
EXTRN DATA (?C_IBP)

            MOV   ?C_IBP,#LOW IBPSTACKTOP
ENDIF

IF XBPSTACK <> 0
EXTRN DATA (?C_XBP)

            MOV   ?C_XBP,#HIGH XBPSTACKTOP
            MOV   ?C_XBP+1,#LOW XBPSTACKTOP
ENDIF

IF PBPSTACK <> 0
EXTRN DATA (?C_PBP)
            MOV   ?C_PBP,#LOW PBPSTACKTOP
ENDIF

            MOV   SP,#?STACK-1
            LJMP  ?C_START

            END
```

Prototype #1 Electronic Schematics

Hand Unit



Base Station



CODE FOR GENERATING A HAND SIZE HISTOGRAM

This MATLAB script generates two hand length histograms, one from data obtained from the Umich:

(http://www.engin.umich.edu/class/ioe433/Biomechanics/Handsurvey.html)

and the other from data obtained through a survey at Dartmouth.

```
%hand_size.m

%plots a histogram of hand length data from a biomechanics study
%at U Michigan
%(http://www.engin.umich.edu/class/ioe433/Biomechanics/Handsurvey.html)

%hand length: distal wrist crease to end of third digit


diary handsize_results;



UMHandLengths = [18.4 16.7 17.1 18.5 17.4 18 18.3 18.6 16.8 16.5 18.7 ...
                 18.5 20.1 20.1 20 19.8 18.9 18.2 19.8 19 19.5 19.4 ...
                 16.8 18 19.5 16 17.5 21.2 20.2 20.4 19.1 ...
                 19.1 20.5 19.5 17.6 19.2 18.4 18.1];

Bins = [10:25];

UMsd = std(UMHandLengths);
UMmed =median(UMHandLengths);
disp(['UMich Study Median:', num2str(UMmed)]);
disp(['UMich Study Standard Deviation:', num2str(UMsd)]);



subplot(2,1,1)
plot(0,0);
hist(UMHandLengths, Bins)

hist_A = gca;
title('Histogram of Hand Lengths -- UMichigan Study Data');
xlabel('Hand Length from Distal Wrist Crease to 3rd Finger Tip (cm)');
ylabel('People (Test Group Size: 38)');
```

```
DCHandLengths = [19 20 19.7 15.5 18.8 18 20.4 19 20.5 21 17 16.5 ...
                 17.5 19 19 19 18.5 20 21];

Bins = [10:25];



DCsd = std(DCHandLengths);
DCmed =median(DCHandLengths);
disp(['Dartmouth Study Median:', num2str(DCmed)]);
disp(['Dartmouth Study Standard Deviation:', num2str(DCsd)]);


subplot(2,1,2)
gca = plot(0,0);
hist(DCHandLengths, Bins)

hist_A = gca;
title('Histogram of Hand Lengths -- Dartmouth Study Data');
xlabel('Hand Length from Distal Wrist Crease to 3rd Finger Tip (cm)');
ylabel('People (Test Group Size: 19)');



diary off;
```

Weight Survey

What is the maximum acceptable weight, in number of quarters, for this one handed typing solution?

_____

Maximum Allowable Weight Survey		
Name	**Weight (in Quarters)**	Weight (g)
Scott Milne	56	336
Shayam	44	264
Lauren Newton	36	216
StephenLee	44	264
Mitch Yashiro	74	444
Leah Platenik	56	336
Kate Turpin	56	336
Joe Brown	36	216
Will Nessle	43	258
		_____
		ave: 297g

FIRMWARE -- Final Prototype

The following programs were written/adapted for driving the 8052 microprocessor and accessing PC AT Keyboard port.

pckey.c

```
/*
PCKEY.C
Krispin Leydon
Spring 2000


FUNCTIONAL DESCRIPTION:
This program is the controlling software for a one handed "chording" PC
keyboard.
Serial output functions modified from code written by Jim Greene,
Signetics Corporation.
*/



/*Include Directives*/
#include <reg52.h>                          /* define 8052 registers */
#include <stdio.h>                          /* define I/O functions */



/*Defines*/
#define bool bit
#define byte unsigned char



/*Static Variable & Constants*/
static const byte TL0_VAL = 0x66;   /*TL0 & TH0: -922 base10*/
                              /*(to count .001sec)*/
static const byte TH0_VAL = 0xFC;
static const byte TH1_VAL = 0xFD;   /*9600 Baud (with 11.0592MHz Xtal)*/
static const byte TL2_VAL = 0xEE;   /*TL2 & TH2: 65518 base10*/
                              /*(to count .0002 sec; a half kbdclk=25KHz
cycle)*/
static const byte  TH2_VAL = 0xFF;



/*Global Variables & Constants*/
data byte prev_chord = 0;
bdata byte current_chord = 0;
data byte key_code = 0;
data byte settle_time = 150;
```

```
data unsigned short int time_before_repeat = 500;
data unsigned short int repeat_delay_time = 60;
bdata bit time0_up = 0;
bdata bit time2_up = 0;
bdata bit data_bit = 0;
bdata bit parity_bit = 0;
bdata bit shift_state = 0;
bdata bit alt_state = 0;
bdata bit control_state = 0;
bdata bit special_case = 0;
sbit data_in = P3 ^ 2;
sbit data_out = P3 ^ 3;
sbit clk_in = P3 ^ 4;
sbit clk_out = P3 ^5;



/*Prototypes*/
byte inverse_of_P1(); /*Returns the Port 1 byte -- all bits complemented*/
bool chord_is_changing (); /*Returns 1 when a new chord begins/ends*/
bool settled (); /*Returns 1 when a chord is maintained for x msec*/
void auto_repeat (); /*Wait to see if "typematic" action is necessary*/
void send_make (byte tomake); /*Sends make code to PC AT bus*/
void send_break (byte tobreak); /*Sends break code to PC AT bus*/
void send_byte (byte tosend); /*Sends byte to PC AT bus*/
void send_bit (); /*Sends data_bit to PC AT bus*/
void delay(byte msec_delay); /*creates a delay*/
byte translate (byte chord); /*Maps chords to chars*/



/*TIMER0_ISR -- Interrupt service routine*/
/*Fires each millisecond, then sets flag "time0_up" to 1*/
/*timer is turned on and flag reset from within other functions.*/
void timer0_isr (void) interrupt 1
      {
      /*Set flag*/
      time0_up = 1;

      /*Turn timer 0 off*/
      TR0 = 0;

      /*Reload Timer0 Value*/
      TL0 = TL0_VAL;
      TH0 = TH0_VAL;

      return;
      }



/*TIMER2_ISR -- Interrupt service routine*/
/*Fires each .0002 sec; a half kbdclk=25KHz cycle*/
/*then sets flag "time2_up" to 1*/
/*timer is turned on and flag reset from within other functions.*/
void timer2_isr (void) interrupt 5
      {
```

```
      /*Set flag*/
      time2_up = 1;
      TF2 = 0;
      EXF2 = 0;

      /*Turn timer 2 off*/
      TR2 = 0;

      /*Reload Timer0 Value*/
      TL2 = TL2_VAL;
      TH2 = TH2_VAL;

      return;
      }



/*MAIN*/
void main ()
      {
      /*Initialize 8051 serial I/O stuff*/
      TMOD = 0x21;
      T2CON = 0x41;
      TL0 = TL0_VAL;
      TH0 = TH0_VAL;
      TH1 = TH1_VAL;
      TL2 = TL2_VAL;
      TH2 = TH2_VAL;
      SCON = 0x5a;
      TCON = 0xc0;
      PT0 = 0;   /*Set timer 0 interrupt to low priority*/
      ET0 = 1;   /*Enable timer 0 interrupt*/
      PT2 = 0;   /*Set timer 2 interrupt to low priority*/
      ET2 = 1;   /*Enable timer 2 interrupt*/
      EA = 1;    /*Global interrupt enable*/


      /*Initialize Other 8051 stuff*/
      P1 = 0xFF;


      /*Initial Prompt*/
        printf("\nType!\n") ;


      /*main (infinite) loop*/
      while(1)
            {
            /*Write All 1s to Port1 Input Bits -- if any are 0 when read,
means input happened*/
            P1 = 0xFF;

            /*Test to see if user has begun to enter a new chord*/
            if (chord_is_changing())
                  {
                        /*New Chord Started:*/
                        /*Wait until the value of P1 settles.*/
```

```
                  /* (Wait until the value of P1 doesn't
                        /* change for a period of x msec).*/

                        while( !settled() )
                              {
                              }

                  /*Send character corresponding to chord to*/
                  /*PC.  If chord is held for a period*/
                  /*of y msec, start repeating the chord (typmatic
                  action--yeah!).*/
                        if (  current_chord != 0x00 &&
                        current_chord != 0x40 &&
                        current_chord != 0x80 &&
                        current_chord != 0xC0    )
                              {
                        /*Send chord to dumb terminal*/
                        printf("\n");
                        printf("%bx", current_chord);
                        printf("\n");
                        /*putchar(current_chord);*/

                        /*Send make and break codes*/
                            key_code = translate(current_chord);
                        send_make(key_code);
                              send_break(key_code);

                        /*Maybe repeat*/
                        auto_repeat();
                        }
                  }
            }
      }


/*INVERSE_OF_P1*/
byte inverse_of_P1 ()
      {
      bdata byte result;

      /*Flip all but the top 2 bits*/
      result = ((P1 ^ 0x3F) ^ 0x00);

      return(result);
      }



/*CHORD_IS_CHANGING*/
bool chord_is_changing ()
      {
      if ( current_chord != inverse_of_P1() )
            {
            prev_chord = current_chord;
            current_chord =  (inverse_of_P1());
            return (1);
```

```
            }

          else
              return (0);
      }



/*SETTLED*/
bool settled ()
      {
      data byte i;

      for ( i = 1; i < settle_time; i++)
            {
            time0_up = 0;
            TR0 = 1;   /*Start timer 0*/
            while (!time0_up)
                  {
                  if (chord_is_changing ())
                        {
                              /*Chord hasn't settled.*/

                        /*Turn timer 0 off*/
                        TR0 = 0;

                        /*Reload Timer0 Value*/
                              TL0 = TL0_VAL;
                        TH0 = TH0_VAL;

                        return(0);
                        }
                  }
            }
      /*Chord has settled!*/
      return(1);
      }



/*AUTO_REPEAT*/
void auto_repeat ()
      {
      data unsigned short int i;       /*an iterator variable*/

      /*Wait for chord to be held for a sufficient ammount of time*/
      /*Return if chord changes during this wait period*/
      for ( i = 1; i < time_before_repeat; i++)
            {

            time0_up = 0;
            TR0 = 1;   /*Start timer 0*/

            while (!time0_up)
                  {
                  if ( current_chord != inverse_of_P1() )
                        {
```

```
                        /*Turn timer 0 off*/
                  TR0 = 0;

                  /*Reload Timer0 Value*/
                        TL0 = TL0_VAL;
                  TH0 = TH0_VAL;

                  return;
                  }
            }
          }

/*Chord has been held a while; auto-repeat character*/
while ( current_chord == inverse_of_P1() &&
      current_chord != 0x00 &&
      current_chord != 0x40 &&
      current_chord != 0x80 &&
      current_chord != 0xC0    )

      {
      /*Send chord to dumb terminal*/
      printf("\n");
      printf("%bx", current_chord);
      printf("\n");

      /*Send make and break codes*/
      key_code = translate(current_chord);
      send_make(key_code);
            send_break(key_code);

      /*Delay after each autorepeat*/
      for ( i = 1; i < repeat_delay_time; i++)
            {

            time0_up = 0;
            TR0 = 1;   /*Start timer 0*/

            while (!time0_up)
                  {
                  if (current_chord != inverse_of_P1())
                              {
                        /*Turn timer 0 off*/
                              TR0 = 0;

                              /*Reload Timer0 Value*/
                                    TL0 = TL0_VAL;
                              TH0 = TH0_VAL;

                              return;
                              }
                  }
            }
      }

return;
}
```

```
/*SEND_MAKE*/
void send_make (byte tomake)
      {

      /*Special Cases -- sending is handled within translate function*/
      if (special_case || key_code == 0x00)
            return;

      /*Normal Cases*/
      if (shift_state)
            send_byte(0x12);  /*Make <shift>*/

      if (control_state)
            send_byte(0x14);  /*Make <cntrl>*/

      if (alt_state)
            send_byte(0x11);  /*Make <alt>*/

      send_byte(tomake);      /*Make 'char'*/

      return;
      }



/*SEND BREAK*/
void send_break (byte tobreak)
      {
      /*Special Cases -- sending is handled within the translate
function*/
      if (special_case || key_code == 0x00)
            return;

      /*Normal Cases*/
      send_byte(0xF0);            /*Break 'char' part1*/
      send_byte(tobreak);         /*Break 'char' part2*/

      if (alt_state)
            {
            send_byte(0xF0);    /*Break <alt> part1*/
            send_byte(0x11);    /*Break <alt> part2*/
                  }

            if (control_state)
            {
            send_byte(0xF0);    /*Break <control> part1*/
            send_byte(0x14);    /*Break <control> part2*/

                  }

      if(shift_state)
            {
            send_byte(0xF0);        /*Break <shift> part1*/
                  send_byte(0x12);  /*Break <shift> part2*/
                  }
```

```
      }


/*SEND_BYTE*/
void send_byte (byte to_send)
      {
      byte i;            /*an iterator variable*/
      bdata bit done;  /*a loop exit flag*/
      byte to_send_copy; /*a working copy of byte to be sent (it gets
destroyed)*/

      /*Make a working copy of byte to send*/
      to_send_copy = to_send;

      /*Determine parity bit necessary for odd parity.
      (Even parity bit for accumulator contents is given in PSW.0.
      Complement this to get odd parity bit)*/
      ACC = to_send_copy;
      parity_bit = (((PSW ^ 0) ^ 1) ^ 0);

      /*Loop until both clock and data lines are high (a hack)*/
      done = 0;
      while (!done)
            {
            clk_in = 1;
            data_in = 1;
            if (clk_in == 1 && data_in == 1)
                  done = 1;
            }

      /*Send start bit */
      data_bit = 0;
      send_bit();


      /*Send to_send_copy, starting with LSB, ending with MSB*/
      for (i=0; i < 8; i++)
            {
            /*Get LSB of to_send_copy (the data_bit)*/
            data_bit =  to_send_copy & 0x01;

            /*Send data bit */
            send_bit();

            /*Right-shift to_send_copy once*/
            to_send_copy = to_send_copy / 0x02;
            }

      /*Send parity bit */
      data_bit = parity_bit;
      send_bit();

      /*Send stop bit */
      data_bit = 1;
      send_bit();
```

```
      /*Raise Clock and Data lines*/
      clk_out = 1;
      data_out = 1;

      /*Wait a bit*/
      delay(1);
      }



/*SEND_BIT*/
void send_bit ()
      {
      /*Put data_bit on the data line*/
      data_out = data_bit;

      /*Wait half a clock period*/
      time2_up = 0;
      TR2 = 1;   /*Start timer 2*/
      while (!time2_up)
            {
            }

      /*Create falling edge*/
      clk_out = 0;

      /*Wait half a clock period*/
      time2_up = 0;
      TR2 = 1;   /*Start timer 2*/
      while (!time2_up)
            {
            }

      /*Create rising edge*/
      clk_out = 1;
      }



/*DELAY*/
void delay(byte msec_delay)
        {
        data byte i;

        for ( i = 1; i < msec_delay; i++)
            {
            time0_up = 0;
            TR0 = 1;   /*Start timer 0*/

            while (!time0_up)
                        {
                        }
            }
      }
```

```
/*TRANSLATE*/
byte translate (byte chord)
      {
      byte make_code = 0x00;
      control_state = 0;
      shift_state = 0;
      alt_state = 0;
      special_case = 0;


      /*Letter Key Codes*/
      if (chord==0x14) { shift_state = 0; make_code = 0x1C; }     /*a*/
      if (chord==0x54) { shift_state = 1; make_code = 0x1C; }     /*A*/

      if (chord==0x17) { shift_state = 0; make_code = 0x32; }     /*b*/
      if (chord==0x57) { shift_state = 1; make_code = 0x32; }     /*B*/
      if (chord==0x26) { shift_state = 0; make_code = 0x21; }     /*c*/
      if (chord==0x66) { shift_state = 1; make_code = 0x21; }     /*C*/
      if (chord==0x1c) { shift_state = 0; make_code = 0x23; }     /*d*/
      if (chord==0x5C) { shift_state = 1; make_code = 0x23; }     /*D*/
      if (chord==0x03) { shift_state = 0; make_code = 0x24; }     /*e*/
      if (chord==0x43) { shift_state = 1; make_code = 0x24; }     /*E*/
      if (chord==0x20) { shift_state = 0; make_code = 0x2B; }     /*f*/
      if (chord==0x60) { shift_state = 1; make_code = 0x2B; }     /*F*/
      if (chord==0x31) { shift_state = 0; make_code = 0x34; }     /*g*/
      if (chord==0x71) { shift_state = 1; make_code = 0x34; }     /*G*/
      if (chord==0x12) { shift_state = 0; make_code = 0x33; }     /*h*/
      if (chord==0x52) { shift_state = 1; make_code = 0x33; }     /*H*/
      if (chord==0x15) { shift_state = 0; make_code = 0x43; }     /*i*/
      if (chord==0x55) { shift_state = 1; make_code = 0x43; }     /*I*/
      if (chord==0x25) { shift_state = 0; make_code = 0x3B; }     /*j*/
      if (chord==0x65) { shift_state = 1; make_code = 0x3B; }     /*J*/
      if (chord==0x09) { shift_state = 0; make_code = 0x42; }     /*k*/
      if (chord==0x49) { shift_state = 1; make_code = 0x42; }     /*K*/
      if (chord==0x05) { shift_state = 0; make_code = 0x4B; }     /*l*/
      if (chord==0x45) { shift_state = 1; make_code = 0x4B; }     /*L*/
      if (chord==0x23) { shift_state = 0; make_code = 0x3A; }     /*m*/
      if (chord==0x63) { shift_state = 1; make_code = 0x3A; }     /*M*/
      if (chord==0x24) { shift_state = 0; make_code = 0x31; }     /*n*/
      if (chord==0x64) { shift_state = 1; make_code = 0x31; }     /*N*/
      if (chord==0x10) { shift_state = 0; make_code = 0x44; }     /*o*/
      if (chord==0x50) { shift_state = 1; make_code = 0x44; }     /*O*/
      if (chord==0x13) { shift_state = 0; make_code = 0x4D; }     /*p*/
      if (chord==0x53) { shift_state = 1; make_code = 0x4D; }     /*P*/
      if (chord==0x27) { shift_state = 0; make_code = 0x15; }     /*q*/
      if (chord==0x67) { shift_state = 1; make_code = 0x15; }     /*Q*/
      if (chord==0x30) { shift_state = 0; make_code = 0x2D; }     /*r*/
      if (chord==0x70) { shift_state = 1; make_code = 0x2D; }     /*R*/
      if (chord==0x11) { shift_state = 0; make_code = 0x1B; }     /*s*/
      if (chord==0x51) { shift_state = 1; make_code = 0x1B; }     /*S*/
      if (chord==0x0a) { shift_state = 0; make_code = 0x2C; }     /*t*/
      if (chord==0x4a) { shift_state = 1; make_code = 0x2C; }     /*T*/
      if (chord==0x22) { shift_state = 0; make_code = 0x3C; }     /*u*/
      if (chord==0x62) { shift_state = 1; make_code = 0x3C; }     /*U*/
      if (chord==0x16) { shift_state = 0; make_code = 0x2A; }     /*v*/
      if (chord==0x56) { shift_state = 1; make_code = 0x2A; }     /*V*/
      if (chord==0x07) { shift_state = 0; make_code = 0x1D; }     /*w*/
```

```
if (chord==0x47) { shift_state = 1; make_code = 0x1D; }     /*W*/
if (chord==0x0f) { shift_state = 0; make_code = 0x22; }     /*x*/
if (chord==0x4f) { shift_state = 1; make_code = 0x22; }     /*X*/
if (chord==0x2a) { shift_state = 0; make_code = 0x35; }     /*y*/
if (chord==0x6a) { shift_state = 1; make_code = 0x35; }     /*Y*/
if (chord==0x1b) { shift_state = 0; make_code = 0x1A; }     /*z*/
if (chord==0x5b) { shift_state = 1; make_code = 0x1A; }     /*Z*/

/*Number Key Codes*/
if (chord==0xa0) { shift_state = 0; make_code = 0x16; }     /*1*/
if (chord==0xB0) { shift_state = 0; make_code = 0x1E; }     /*2*/
if (chord==0x9C) { shift_state = 0; make_code = 0x26; }     /*3*/
if (chord==0x85) { shift_state = 0; make_code = 0x25; }     /*4*/
if (chord==0x92) { shift_state = 0; make_code = 0x2E; }     /*5*/
if (chord==0xAA) { shift_state = 0; make_code = 0x36; }     /*6*/
if (chord==0xB1) { shift_state = 0; make_code = 0x3D; }     /*7*/
if (chord==0xA3) { shift_state = 0; make_code = 0x3E; }     /*8*/
if (chord==0xA2) { shift_state = 0; make_code = 0x46; }     /*9*/
if (chord==0x90) { shift_state = 0; make_code = 0x45; }     /*0*/

/*Other Symbol Key Codes*/
if (chord==0x0b || chord==0x4b)
      { shift_state = 1; make_code = 0x16; }                /*!*/
if (chord==0x94) { shift_state = 1; make_code = 0x1E; }     /*@*/
if (chord==0x9b) { shift_state = 1; make_code = 0x26; }     /*#*/
if (chord==0x91) { shift_state = 1; make_code = 0x25; }     /*$*/
if (chord==0x8F) { shift_state = 1; make_code = 0x2E; }     /*%*/
if (chord==0xa7) { shift_state = 1; make_code = 0x36; }     /*^*/
if (chord==0x99) { shift_state = 1; make_code = 0x3D; }     /*&*/
if (chord==0x87) { shift_state = 1; make_code = 0x3E; }     /***/
if (chord==0x2c || chord==0x6c || chord==0xac)
      { shift_state = 1; make_code = 0x46; }                /*(*/
if (chord==0x3c || chord==0x7c || chord==0xbc)
      { shift_state = 1; make_code = 0x45; }                /*)*/
if (chord==0x0d || chord==0x4d || chord==0x8d)
      { shift_state = 0; make_code = 0x4E; }                /*-*/
if (chord==0x8a) { shift_state = 0; make_code = 0x55; }     /*=*/
if (chord==0xa9) { shift_state = 0; make_code = 0x0E; }     /*`*/
if (chord==0x89) { shift_state = 0; make_code = 0x54; }     /*[*/
if (chord==0x96) { shift_state = 0; make_code = 0x5B; }     /*]*/
if (chord==0x8b) { shift_state = 0; make_code = 0x5D; }     /*\*/
if (chord==0x1d || chord==0x5d)
      { shift_state = 0; make_code = 0x4C; }                /*;*/
if (chord==0x19 || chord==0x59)
      { shift_state = 0; make_code = 0x52; }                /*"*/
if (chord==0x29 || chord==0x69)
      { shift_state = 0; make_code = 0x41; }                /*,*/
if (chord==0x1a || chord==0x5a)
       { shift_state = 0; make_code = 0x49; }               /*.*/
if (chord==0xa4){ shift_state = 0; make_code = 0x4A; }      /*/*/
if (chord==0xa6) { shift_state = 1; make_code = 0x4E; }     /*_*/
if (chord==0x93) { shift_state = 1; make_code = 0x55; }     /*+*/
if (chord==0x9a) { shift_state = 1; make_code = 0x0E; }     /*~*/
if (chord==0xa5) { shift_state = 1; make_code = 0x54; }     /*{*/
if (chord==0xab) { shift_state = 1; make_code = 0x5B; }     /*}*/
if (chord==0x95) { shift_state = 1; make_code = 0x5D; }     /*|*/
if (chord==0x34 || chord==0x74)
```

```
       { shift_state = 1; make_code = 0x4C; }              /*:*/
if (chord==0x2b || chord==0x6b)
      { shift_state = 1; make_code = 0x52; }               /*'*/
if (chord==0x97) { shift_state = 1; make_code = 0x41; }    /*<*/
if (chord==0x83) { shift_state = 1; make_code = 0x49; }    /*>*/
if (chord==0x21 || chord==0x61)
      { shift_state = 1; make_code = 0x4A; }               /*?*/
if (chord==0x02 || chord==0x42 || chord==0x82 || chord==0xc2)
       { shift_state = 0; make_code = 0x29; }        /*<SPACE>*/
if (chord==0xd4) { shift_state = 0; make_code = 0x11; }
/*<ALT>*/
if (chord==0x28 || chord==0x68 || chord==0xa8 || chord==0xe8)
      { shift_state = 0; make_code = 0x0D; }
/*<TAB>*/
if (chord==0xe8) { shift_state = 1; make_code = 0x0D; }
/*<SHIFT-TAB>*/
if (chord==0x01 || chord==0x41 || chord==0x81 || chord==0xC1)
      { shift_state = 0; make_code = 0x5A; }
/*<ENTER/RETURN>*/
if (chord==0x04 || chord==0x44 || chord==0x84 || chord==0xc4)
      { shift_state = 0; make_code = 0x66; }
/*<BACKSPACE/DELETE>*/
if (chord==0xc3) { shift_state = 0; make_code = 0x76; }
/*<ESC>*/
if (chord==0x18 || chord==0x58 || chord==0x98)
                                          /*<UP ARROW>*/
      {
      special_case = 1;
      send_byte(0xe0);
      send_byte(0x75);
            send_byte(0xe0);
            send_byte(0xf0);
            send_byte(0x75);
            }
if (chord==0x06 || chord==0x46 || chord==0x86)
                                          /*<DOWN ARROW>*/
      {
      special_case = 1;
      send_byte(0xe0);
      send_byte(0x72);
            send_byte(0xe0);
            send_byte(0xf0);
            send_byte(0x72);
            }
if (chord==0x08 || chord==0x48 || chord==0x88)
                                          /*<LEFT ARROW>*/
      {
      special_case = 1;
        send_byte(0xe0);
      delay(3);
      send_byte(0x6b);
      delay(25);
      send_byte(0xe0);
      delay(3);
      send_byte(0xf0);
      delay(3);
      send_byte(0x6B);
```

```
            }
if (chord==0x0c || chord==0x4c || chord==0x8c)
                                              /*<RIGHT ARROW>*/
      {
      special_case = 1;
        send_byte(0xE0);
      send_byte(0x74);
            send_byte(0xE0);
            send_byte(0xF0);
            send_byte(0x74);
            }
if (chord==0xd8)                               /*<SHIFTED UP ARROW>*/
      {
      special_case = 1;
            send_byte(0x12);
      send_byte(0xE0);
      send_byte(0x75);
            send_byte(0xE0);
      send_byte(0xF0);
      send_byte(0x75);
      send_byte(0xF0);
      send_byte(0x12);
            }
if (chord==0xc6)                              /*<SHIFTED DOWN ARROW>*/
      {
      special_case = 1;
      send_byte(0x12);
      send_byte(0xE0);
      send_byte(0x72);
            send_byte(0xE0);
      send_byte(0xF0);
      send_byte(0x72);
      send_byte(0xF0);
      send_byte(0x12);
            }
if (chord==0xc8)                              /*<SHIFTED LEFT ARROW>*/
      {
      special_case = 1;
        send_byte(0x12);
      send_byte(0xE0);
      send_byte(0x6B);
            send_byte(0xE0);
      send_byte(0xF0);
      send_byte(0x6B);
      send_byte(0xF0);
      send_byte(0x12);
            }
if (chord==0xcc)                              /*<SHIFTED RIGHT ARROW>*/
      {
      special_case = 1;
      send_byte(0x12);
      send_byte(0xE0);
      send_byte(0x74);
            send_byte(0xE0);
      send_byte(0xF0);
      send_byte(0x74);
      send_byte(0xF0);
```

```
            send_byte(0x12);
                  }

      if (chord==0xe6) { shift_state = 0; make_code = 0x21; /*<CONTROL-
C>*/
                  control_state = 1;}
      if (chord==0xcf) { shift_state = 0; make_code = 0x22; /*<CONTROL-
X>*/
                  control_state = 1;}
      if (chord==0xd6) { shift_state = 0; make_code = 0x2a; /*<CONTROL-V*/
                  control_state = 1;}
      if (chord==0xe4) { shift_state = 0; make_code = 0x31; /*<CONTROL-N*/
                  control_state = 1;}
      if (chord==0xd0) { shift_state = 0; make_code = 0x44; /*<CONTROL-O*/
                  control_state = 1;}
      if (chord==0xd3) { shift_state = 0; make_code = 0x4d; /*<CONTROL-P*/
                  control_state = 1;}
      if (chord==0xe7) { shift_state = 0; make_code = 0x15; /*<CONTROL-Q*/
                  control_state = 1;}
      if (chord==0xd1) { shift_state = 0; make_code = 0x1b; /*<CONTROL-S*/
                  control_state = 1;}
      if (chord==0xc7) { shift_state = 0; make_code = 0x1d; /*<CONTROL-W*/
                  control_state = 1;}
            if (chord==0xd7)                              /*<CONTROL-ALT-
DELETE>*/
                  {
                  special_case = 1;
                  send_byte(0x14);
            send_byte(0x11);
            send_byte(0xE0);
                  send_byte(0x71);

            send_byte(0xF0);
            send_byte(0x14);

            send_byte(0xF0);
            send_byte(0x11);

                  send_byte(0xE0);
            send_byte(0xF0);
                  send_byte(0x71);

            delay(20);
            clk_out = 1;
            data_out = 1;
            delay(20);
                  }

      if (chord == 0xd9)
            {                                    /*<'and'>*/
            special_case = 1;

            send_byte(0x1c);/*'a'*/
            send_byte(0xF0);
            send_byte(0x1c);

            send_byte(0x31);/*'n'*/
```

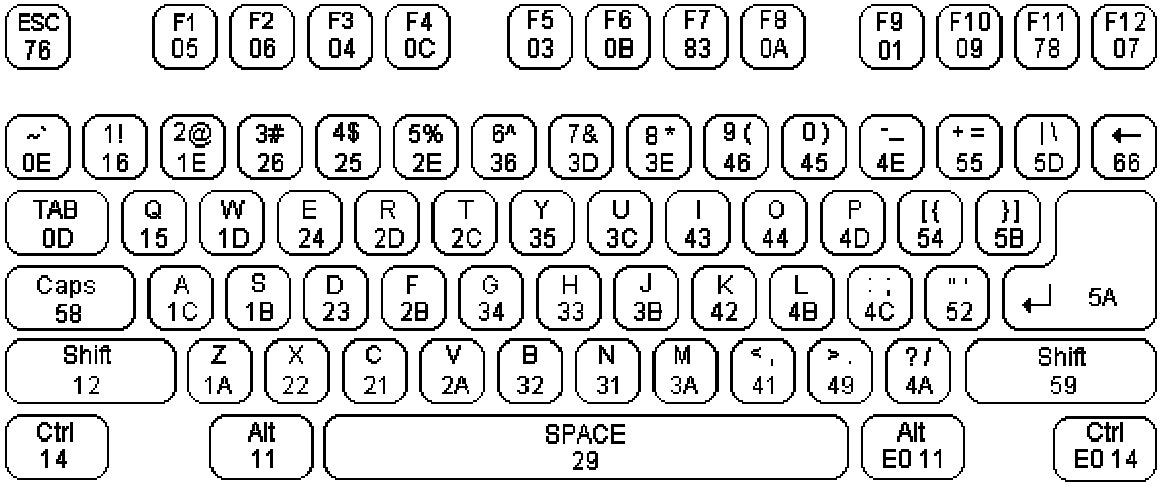
```
            send_byte(0xF0);
            send_byte(0x31);

            send_byte(0x23);/*'d'*/
            send_byte(0xF0);
            send_byte(0x23);
            }

      delay(1);

      return(make_code);
      }
```

Startup.a51

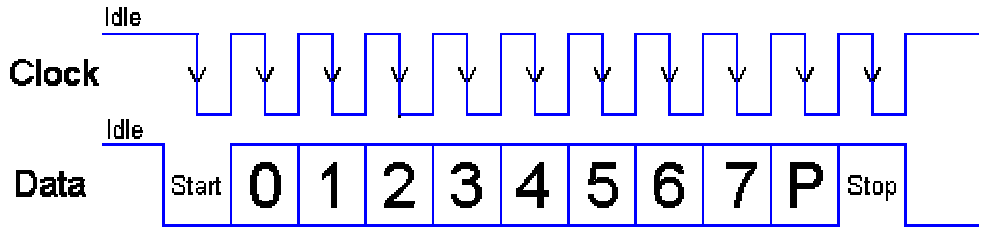
This listing is already included in the code listing for the prototype 1.

PC-AT Keyboard Communications Protocol

"Make/Break" Key Codes



Data Transmission

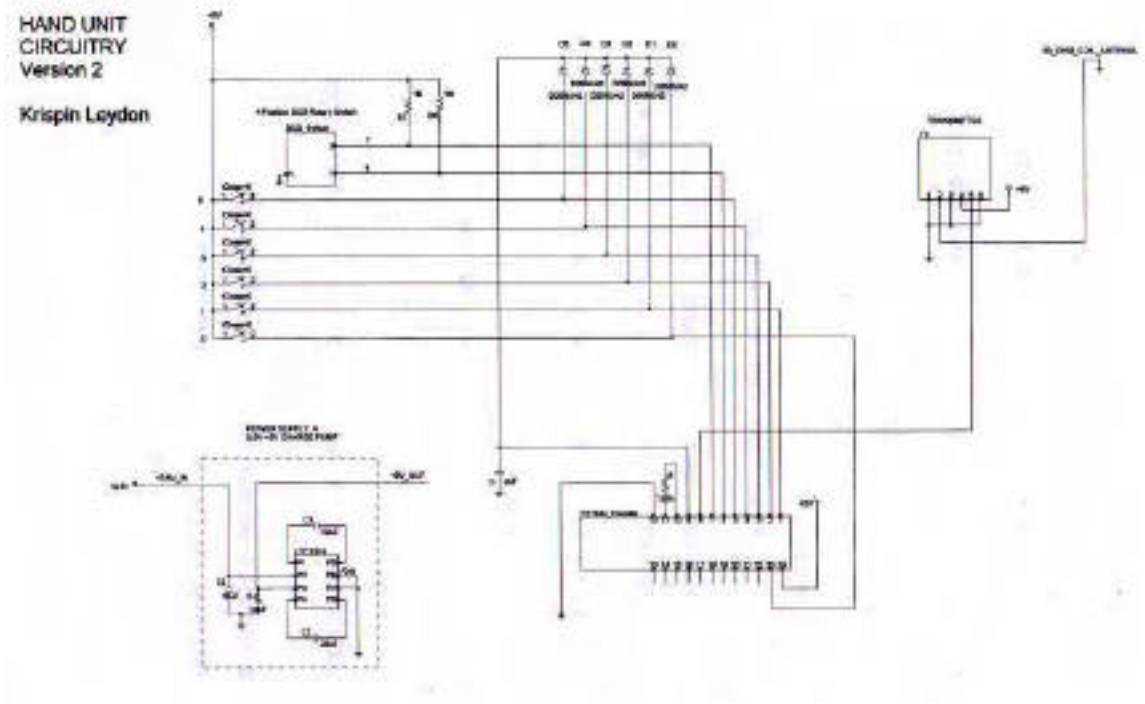


Images reproduced from the website:
http://www.beyondlogic.org/keyboard/keybrd.htm

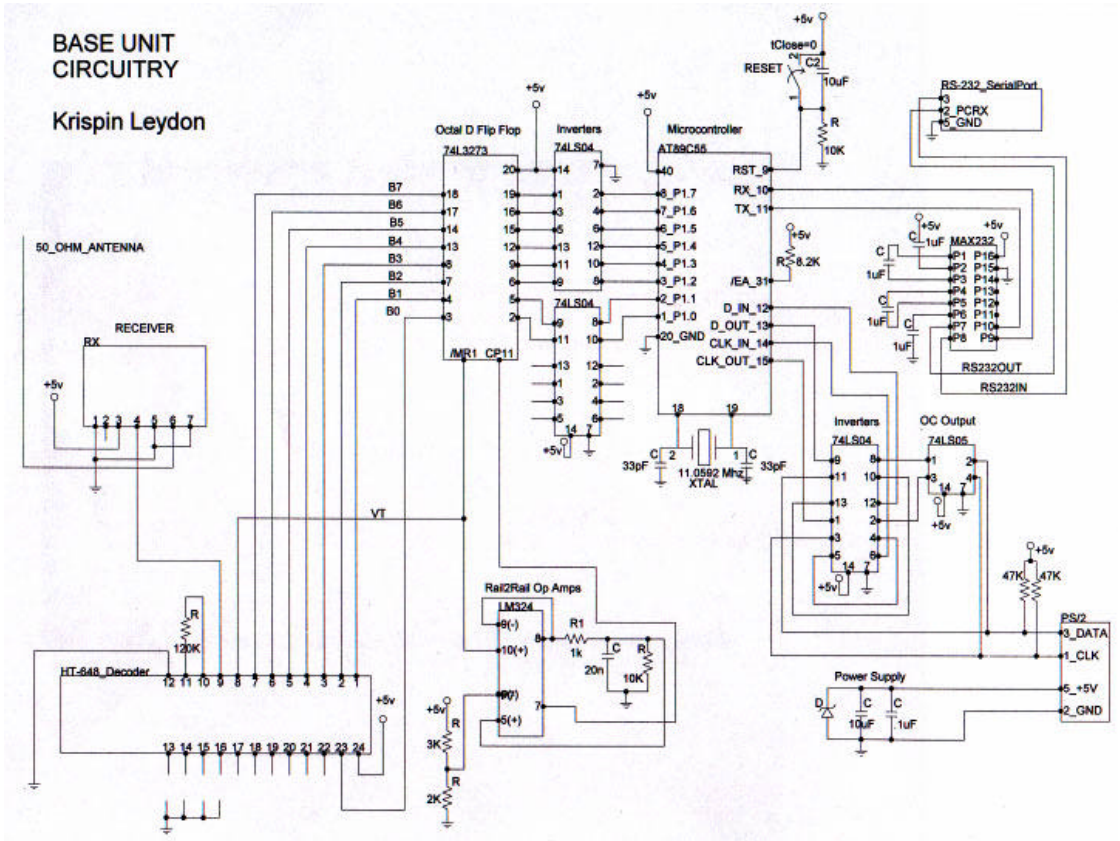
Craig Peacock

Prototype #2 Electronic Schematics

Hand Unit



Base Station



CODE FOR TESTING CHORD "INTUITIVENESS"

Written by Michael Fromberger

This Java program displays a picture of a hand, with botton contact locations marked as circles. The program randomly cycles through pictures of all the possible button combinations, and prompts the user to copy these combinations using the chording keyboard. For each button combination (finger "chord"), the program logs the following information:

1) Time taken to press (or skip) the chord.
2) Whether the chord was completed or skipped.
3) The number of error chords pressed before a correct press or a skip.

"The program requires Java 1.2.x
or higher, and a copy of the CommAPI from
http://java.sun.com/,
so that it can talk to the COM ports on your PC" (Mike).

Fingers.java

```
/*
        Fingers.java

        Main driver for the chording keyboard tester

        Shuffle all the possible combinations of keys, display them k times (for some
        small constant k).  Record how long until they got it right, target keystroke,
        and number of misses (i.e., how many false hits they got).

        At startup, get the person's name, record that in the output file.

        Give feedback after every time they press a key, of whether they got it right.
        "right" or "try again"
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.applet.*;
import java.io.*;
```

```
public class Fingers
{
	private static final int	KEY_MASK = 0x3F;
	private static final int	DATA_POINTS = 63;	//	How many data points to
test
	private static final String	IMG_FILE = "hand.gif"; //	File containing background
image
	private static final String FRAME_TITLE =		//	Title of main frame
						"Chording Keyboard Tester";
	private static final int	PAUSE_TIME = 1500;	//	Pause interval between
prompts
	private static final int	INTERTEST_PAUSE_TIME = 2000;  //	Pause interval
btwn tests

	private  static int[]		testVector;			//	Values to test the
user on
	private  static int			nextVector;			//	Index of
the next value to test
	private  static String		userName;			//	Subject we are
testing now
	private  static DataPoint	currentPoint;		//	Point we're gathering now
	private  static DataPoint[]	dataPoints;			//	The data collected
so far

	private static FingerWindow	mainWindow;
	private static boolean		isRunning = false;
	private static boolean		isSkipping = false;

	private static SerialHandler	dataReader;

	public static void main(String[] args)
	{

		//	Get the user's name
		try {
			userName = getUserName();
		} catch(UserCancelledException e) {
			System.exit(0);
		}

		//	Create the main window
		mainWindow = new FingerWindow(FRAME_TITLE, IMG_FILE,
								new
FingerButtonListener());

		//	Create the serial port handler
		dataReader = new SerialHandler("COM1");

		//	Let's get this whole thing started, shall we?
		mainWindow.show();

		//	Set up the data set to test the user with
		testVector = new int[DATA_POINTS];
		makePermutation(testVector);
		nextVector = 0;
		dataPoints = new DataPoint[DATA_POINTS];

		//	Wait until someone hits the start button
		while(!isRunning) {
			try {
				Thread.sleep(100);
			} catch(InterruptedException e) { }
		}

		//	Set the buttons for the running mode
		mainWindow.start();

		//	Loop until done, handling skips and all
		while(isRunning) {
			//	Display the next test to be run
			mainWindow.setCounter(DATA_POINTS - nextVector);
```

```
                        mainWindow.setMessage("Ready...");
                        pause(PAUSE_TIME);
                        dataReader.flushData();       //     Ignore held over stuff

                        mainWindow.setMessage("Go!");
                        currentPoint = new DataPoint(testVector[nextVector], 0);
                        mainWindow.setLights(testVector[nextVector]);
                        currentPoint.startTiming();

                        //     While we're running, and haven't been told to skip...
                        while(isRunning && !isSkipping) {

                                //     Check if there's anything pending on the input
                                if(dataReader.hasData()) {
                                        //     If so, get it
                                        byte[]  data = dataReader.getData();

                                        //     Convert the last thing that we caught being
entered
                                        int             key = (int)data[data.length - 1];

                                        //     See if it's the right key, and update display
                                        if(currentPoint.checkValue(key & KEY_MASK)) {
                                                mainWindow.setMessage("Correct!");
                                                break;
                                        } else {
                                                mainWindow.setMessage("Sorry, try again!");
                                        }
                                }
                                pause(5);       //     Ugh, I know this sucks, but...
                        }

                        //     Handle a skip, if it occurred
                        if(isSkipping) {
                                currentPoint.cancel();
                                mainWindow.setMessage("Skipping this one");
                                isSkipping = false;
                        }

                        if(isRunning) {
                                pause(INTERTEST_PAUSE_TIME);
                                mainWindow.setLights(0);
                                dataPoints[nextVector] = currentPoint;
                                ++nextVector;
                        }

                        //     When we're done...
                        if(nextVector >= dataPoints.length)
                                isRunning = false;
                }
                System.out.println("Done with testing loop");

                String  fileName = userName.toLowerCase() + ".txt";
                try {
                        mainWindow.setMessage("Saving data...");
                        PrintWriter    outFile = new PrintWriter(new BufferedWriter(new
FileWriter(fileName)));
                        outFile.println("% Data for " + userName);
                        outFile.println("% " + nextVector + " data points collected");
                        if(nextVector == dataPoints.length)
                                outFile.println("% Data collection complete");
                        else
                                outFile.println("% Data collection interrupted");

                        outFile.println("%");
                        outFile.println("% Key\tDone\tTime\tErrors\n%");
                        for(int ix = 0; ix < nextVector; ix++)
                                outFile.println(dataPoints[ix]);

                        outFile.close();
                } catch(IOException e) {
```

```
                        try {
                                JOptionPane.showMessageDialog(null,
                                        "Couldn't open file '" + fileName + "' for writing!");
                        } catch(NullPointerException npe) {
                                //      ignore...
                        }
                }

                mainWindow.setMessage("Goodbye!");
                mainWindow.hide();
                System.exit(0);

        }       // end of main()

        //      Use a an input dialog to get the user's name...
        public static String getUserName() throws UserCancelledException
        {
                String  output = null;

                while(true) {
                        output = JOptionPane.showInputDialog("Please enter your name");

                        if(output.equals("")) {
                                try {
                                        JOptionPane.showMessageDialog(null,
                                                                                "I'm
sorry, but I need something besides "
                                                                              + "blank
for your name!");
                                } catch(NullPointerException e) {
                                        //      ignore...
                                }
                        } else if(output == null) {
                                throw new UserCancelledException();
                        } else {
                                return output;
                        }
                }

        }       // end of getUserName()

        //      Make an array of integers into a permutation of the values from
        //      one to the length of the array.  Uses Knuth's classic shuffle.
        public static void makePermutation(int[] array)
        {
                for(int ix = 0; ix < array.length; ix++)
                        array[ix] = ix + 1;

                for(int ix = array.length - 1; ix >= 0; ix--) {
                        int which = (int)Math.floor(Math.random() * ix);

                        int tmp = array[which];
                        array[which] = array[ix];
                        array[ix] = tmp;
                }

        }       // end of makePermutation()

        //      This is where all button pressees in the main window are handled
        private static class FingerButtonListener implements ActionListener
        {
                public void actionPerformed(ActionEvent evt)
                {
                        JButton btn = (JButton)evt.getSource();
                        String  text = btn.getText();

                        System.out.println("Clicked: " + text);
                        if(text.equals("Start")) {
                                synchronized(mainWindow) { isRunning = true; }
                        } else if(text.equals("Skip")) {
                                synchronized(mainWindow) { isSkipping = true; }
```

```
                        } else {       //     Quit button
                                synchronized(mainWindow) { isRunning = false; }
                        }
                }

        }       //      end of class FingerActionListener

        //      This is thrown by the getUserName() function, above...
        private static class UserCancelledException extends Exception
        {
                //      Doesn't contain anything...

        }       //      end of class UserCancelledException

        private static void pause(int ms)
        {
                try {
                        Thread.sleep(ms);
                } catch(InterruptedException e) { }
        }
} // end of class Fingers
```

DataPoint.java

```
/*
      DataPoint.java

      A single data element collected from the user, encapsulated in a
package

      by Michael J. Fromberger <sting@linguist.dartmouth.edu>
      Copyright (C) 2000 The Trustees of Dartmouth College
 */

public class DataPoint
{
      private int       target;           //    What element we were
trying to test
      private long      startTime;  //    When we started timing
      private long      finishTime; //    When we finished timing
      private boolean   cancelled;  //    Did the user abort this data
point?
      private int       errors;           //    How many errors did the
user make?

      public DataPoint(int target, long startTime)
      {
            this.target = target;
            this.startTime = startTime;
            this.finishTime = 0;
            this.cancelled = false;
            this.errors = 0;

      }     //    end of DataPoint() constructor

      public DataPoint()
```

```
	{
		this(0, 0);

	}	//	end of DataPoint() constructor

	public void setTarget(int target) { this.target = target; }
	public void startTiming()  { startTime = System.currentTimeMillis();
}
	public void finishTiming() { finishTime =
System.currentTimeMillis(); }
	public void cancel() { finishTiming(); cancelled = true; }
	public void error()  { errors++; }

	//	Check a candidate value, and stop timing if it's a good one;
record an
	//	error if it is not
	public boolean checkValue(int v)
	{
		if(v == target) {
			finishTiming();
			return true;
		} else {
			error();
			return false;
		}
	}

	public String toString()
	{
		return		Integer.toHexString(target) + "\t" +
				(cancelled ? "0" : "1") + "\t" +
				(finishTime - startTime) + "\t" +
				errors;

	}	//	end of toString()

	//	Test driver for the DataPoint class
	public static void main(String[] args)
	{
		DataPoint one = new DataPoint(), two = new DataPoint(101, 0);

		one.startTiming();
		two.startTiming();

		try {
			Thread.sleep(593);
		} catch(InterruptedException e) {
			//	ignore
		}

		one.cancel();
		two.error();
		two.error();
		two.finishTiming();

		System.out.println("one => " + one);
		System.out.println("two => " + two);
```

```
	}	//	end of main()

}	//	end of class DataPoint
```

FingerWindow.java

```
/*
	FingerWindow.java

	Main user interface object for the chording keyboard tester.

	by Michael J. Fromberger <sting@linguist.dartmouth.edu>
	Copyright (C) 2000 The Trustees of Dartmouth College
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class FingerWindow
{
	private static int[] lightx = {	//	x coordinates of the
finger pad lights
		211, 135, 82, 60,  45,  20
	};
	private static int[] lighty = {	//	y coordinates of the
finger pad lights
		 43,  25, 66, 42, 129, 104
	};
	private static final int LED_SIZE = 20;	//	Size of the finger
pad lights (pixels)

	private JFrame		mainFrame;		//	The main
window
	private LEDPanel  lightBoard;		//	Where the lights are
displayed
	private JButton   start, skip, quit;		//	Buttons the
user can press
	private JLabel		feedback;		//	Where the
status goes...
	private	JLabel		countdown;		//
	Countdown marker

	public FingerWindow(String title, String imageFile, ActionListener
buttonMaster)
	{
		//	An LEDPanel is the main display element here
```

```
            lightBoard = new LEDPanel();

            //    Force the image to be loaded, so we can get its size
            Image       pic =
Toolkit.getDefaultToolkit().getImage(imageFile);
            MediaTracker      mt = new MediaTracker(lightBoard);
            mt.addImage(pic, 0);
            try { mt.waitForAll(); } catch(InterruptedException e) { }

            //    Set it to be the background image of the LED panel
            lightBoard.setBackgroundImage(pic);

            //    Plug in the LED's.  This is a little hackish, since the
values are given
            //    computed empirically, but it works for this application
            for(int ix = 0; ix < lightx.length; ix++) {
                  lightBoard.addLED(lightx[ix], lighty[ix], LED_SIZE,
false);
            }

            //    Create the main frame, set its title and background
colour
            mainFrame = new JFrame();
            mainFrame.setTitle(title);
            mainFrame.setBackground(Color.white);

            //    Get out the main content pane, since Swing likes us to
add interface elements
            //    there, instead of to the frame itself
            Container   pane = mainFrame.getContentPane();

            //    Create a panel to hold the input elements, and set its
background colour
            JPanel            bbox = new JPanel();
            bbox.setBackground(Color.white);

            //    Create a box to put the buttons in, aligned horizontally
            bbox.setLayout(new BoxLayout(bbox, BoxLayout.X_AXIS));

            //    Create the buttons, and bind them to the action listener
            start = new JButton("Start");
            skip = new JButton("Skip");
            quit = new JButton("Quit");

            skip.setEnabled(false); //    skip is initially disabled...

            //    Create an action listener to handle the buttons, and
register it with the
            //    buttons.  We'll use the same listener for all the
buttons, in this case
            start.addActionListener(buttonMaster);
            skip.addActionListener(buttonMaster);
            quit.addActionListener(buttonMaster);

            //    Create the label used to give the user feedback about
the state of things
            feedback = new JLabel("xxxxxxxxxxxxxxxxxxxxxxxx");
```

```
            feedback.setForeground(Color.red);
            feedback.setFont(new Font("Serif", 0, 20));

            //    Create the label used to give the user a countdown on
the number of tests
            countdown = new JLabel();
            countdown.setForeground(Color.blue.darker());
            countdown.setFont(new Font("Serif", 0, 20));

            //    Drop the buttons into the panel...
            bbox.add(Box.createHorizontalStrut(3));         //    leave
space around the edge
            bbox.add(start);
            bbox.add(Box.createHorizontalStrut(6));         //    leave
space between the buttons
            bbox.add(skip);
            bbox.add(Box.createHorizontalStrut(6));
            bbox.add(quit);
            bbox.add(Box.createHorizontalStrut(10));  //    leave space
before the label
            bbox.add(countdown);
            bbox.add(Box.createHorizontalStrut(10));  //    Leave space
between labels
            bbox.add(feedback);
            bbox.add(Box.createHorizontalStrut(3));         //    leave
space around the edge

            //    Put the pieces into the frame's content pane
            pane.setBackground(Color.white);
            pane.add(lightBoard, BorderLayout.CENTER);
            pane.add(bbox, BorderLayout.SOUTH);

            //    Oh, and by the way, update the frame for all these

                  outFile.println("%");
                  outFile.println("% Key\tDone\tTime\tErrors\n%");
                  for(int ix = 0; ix < nextVector; ix++)
                        outFile.println(dataPoints[ix]);

                  outFile.close();
            } catch(IOException e) {
                  try {
                        JOptionPane.showMessageDialog(null,
                              "Couldn't open file '" + fileName + "' for
writing!");
                  } catch(NullPointerException npe) {
                        //    ignore...
                  }
            }

            mainWindow.setMessage("Goodbye!");
            mainWindow.hide();
            System.exit(0);

      }     // end of main()
= 1);
                  LED led = lightBoard.getLED(bit);
```

```
                led.setLit(on);
                value >>>= 1;
          }

          lightBoard.repaint();

    }     //    end of setupLights()

    //    Set the message on the feedback label
    public void setMessage(String msg)
    {
          feedback.setText(msg);

    }     //    end of setMessage()

    //    Set the counter
    public void setCounter(int value)
    {
          countdown.setText(Integer.toString(value));
    }

} // end of class FingerWindow
```

LED.java

```
/*
      LED.java

      Defines a simple graphical equivalent of a light-emitting diode,
that can
      be used as a feeback element (like a label, only graphical).

      by Michael J. Fromberger <sting@linguist.dartmouth.edu>
      Copyright (C) 2000 The Trustees of Dartmouth College
 */

import java.awt.*;

public class LED
{
      private final Color     DEFAULT_LIT_COLOR = Color.red;
      private final Color DEFAULT_OUT_COLOR = Color.white;

      private Color     litColor, outColor;     //    Colour when lit up
or blacked out
      private Point     center;                 //    Where the LED
is located (center point)
      private     int         size;             //    The radius of
the LED
```

```
	private boolean lit;			//	Whether it's
currently lit up

	/**
		Construct a new LED with given location, size, and colour
characteristics.

		@param center	The center of where the LED should be drawn
		@param size	The diameter of the LED
		@param litColor	What colour the LED should be when lit up
		@param outColor	What colour the LED should be when not lit
	 */
	public LED(Point center, int size, Color litColor, Color outColor)
	{
		this.litColor = litColor;
		this.outColor = outColor;
		this.center = center;
		this.size = (size + 1) / 2;
		lit = false;

	}

	//	Like the above constructor, except with the location given as
(x, y) values
	public LED(int x, int y, int size, Color lit, Color out)
	{
		this(new Point(x, y), size, lit, out);

	}

	//	Like the above constructor, except with colour values
defaulted
	public LED(Point center, int size)
	{
		this(center, size, Color.black, Color.white);
		litColor = DEFAULT_LIT_COLOR;
		outColor = DEFAULT_OUT_COLOR;

	}

	//	Like the above constructor, except with colour values
defaulted and location
	//	given as (x, y) values
	public LED(int x, int y, int size)
	{
		this(x, y, size, Color.black, Color.white);
		litColor = DEFAULT_LIT_COLOR;
		outColor = DEFAULT_OUT_COLOR;

	}

	//	Various methods for handling the lit-up state of the LED
	public boolean	isLit()	{ return lit;}
	public void	light()		{ lit = true; }
	public void	off()	{ lit = false; }
	public void	toggle()	{ lit = !lit; }
	public void	setLit(boolean to) { lit = to; }
```

```
	//	Drawing the LED
	public void draw(Graphics area)
	{
		if(lit)
			area.setColor(litColor);
		else
			area.setColor(outColor);

		area.fillOval(center.x - size, center.y - size, 2 * size, 2 *
size);

		if(lit)
			area.setColor(litColor.darker());
		else
			area.setColor(Color.black);

		area.drawOval(center.x - size, center.y - size, 2 * size, 2 *
size);
		area.drawOval(center.x - size + 1, center.y - size + 1, 2 *
size - 2, 2 * size - 2);
	}

	//	Test driver
	public static void main(String[] args)
	{
		System.out.println("Test driver for LED class\n");

		Frame window = new Frame("LED Test");
		LED		l1, l2, l3, l4;

		l1 = new LED(20, 20, 20);	//	Default colour LED
		l2 = new LED(50, 20, 20, Color.green, Color.white);
		l3 = new LED(80, 30, 20);
		l4 = new LED(110, 40, 30, Color.blue, Color.white);

		window.pack();
		window.setSize(200, 80);
		window.show();

		try {
			Thread.sleep(1000);

		} catch(InterruptedException e) {
			// ignore
		}

		Graphics area = window.getGraphics();
		l1.light(); l1.draw(area);
		l2.light(); l2.draw(area);
		l3.draw(area);
		l4.light(); l4.draw(area);

		try {
			Thread.sleep(5000);

		} catch(InterruptedException e) {
```

```
                // ignore
            }
            window.dispose(); // Release frame back to the environment
      }
}

// Here there be dragons
```

LEDPanel.java

```
/*
      LEDPanel.java

      A simple panel that displays a collection of LED objects

      by Michael J. Fromberger <sting@linguist.dartmouth.edu>
      Copyright (C) 2000 The Trustees of Dartmouth College

 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LEDPanel extends JPanel
{
      private static final int PADDING = 5;  // How many pixels of padding
around the content

      private Image     bg = null;
      private     LED[] lights;
      private int     numLED = 0;

      //    Create a new, empty LED panel
      public LEDPanel()
      {
            super();

            lights = new LED[4];
      }

      //    Set the background image (if not set, no image is displayed)
      public void setBackgroundImage(Image img)
      {
            bg = img;
```

```
            //    Set the size of the panel to have room for the image,
plus some padding
            int width = bg.getWidth(null);
            int height = bg.getHeight(null);

            setMinimumSize(new Dimension(width + 2 * PADDING, height + 2 *
PADDING));
            setPreferredSize(new Dimension(width + 2 * PADDING, height + 2
* PADDING));
      }

      //    Add another LED to the display
      public void addLED(int x, int y, int size, boolean isLit)
      {
            LED   newLED = new LED(x, y, size);

            newLED.setLit(isLit);
            lights[numLED++] = newLED;

            //    If we filled up the array of lights, grow it
            if(numLED >= lights.length) {
                  LED[] tmp = new LED[2 * lights.length];

                  for(int ix = 0; ix < numLED; ix++)
                        tmp[ix] = lights[ix];

                  lights = tmp;
            }
      }

      //    How many LED's are currently in the set?
      public int countLED() { return numLED; }

      //    Fetch one of the LED's, given its index.  Note that we don't
check
      //    the given index -- if you go out of bounds, you'll get the
usual
      //    ArrayIndexOutOfBounds exception.
      public LED getLED(int which)
      {
            return lights[which];

      }

      //    Override the paint method so that the contents get painted
properly
      public void paintComponent(Graphics page)
      {
            page.clearRect(0, 0, getWidth(), getHeight());

            //    Paint the background image, if there is one
            if(bg != null) {
                  page.drawImage(bg, PADDING, PADDING, null);
            }

            //    Draw in any LED's that are there...
```

```
            for(int ix = 0; ix < numLED; ix++)
                  lights[ix].draw(page);
      }

      //    Simple test driver for the frame...
      public static void main(String[] args) throws InterruptedException
      {
            final int   LED_SIZE = 20;          //    How big are those
lights, anyway?
            final int   BUTTON_ROOM = 25; //    How much extra space for a
button?

            JFrame            frm = new JFrame();
            LEDPanel    myPanel = new LEDPanel();
            Image       myImage =
Toolkit.getDefaultToolkit().getImage("hand.gif");

            //    Force the image to load
            MediaTracker      loader = new MediaTracker(frm);
            loader.addImage(myImage, 0);
            loader.waitForAll();

            //    Now, size the frame so it's big enough
            int h = myImage.getHeight(null);
            int w = myImage.getWidth(null);
            frm.setSize(w + 2 * PADDING, h + 2 * PADDING + BUTTON_ROOM);

            //    Throw in some lights (assorted starting values)
            myPanel.setBackgroundImage(myImage);
            myPanel.addLED(20, 104, LED_SIZE, false);
            myPanel.addLED(45, 129, LED_SIZE, false);
            myPanel.addLED(60, 42, LED_SIZE, false);
            myPanel.addLED(82, 66, LED_SIZE, false);
            myPanel.addLED(135, 25, LED_SIZE, false);
            myPanel.addLED(211, 43, LED_SIZE, false);

            double d = Math.random();
            byte b = (byte)Math.floor(d * 64);
            int    ix = 0;
            while(b != 0) {
                  LED tmp = myPanel.getLED(ix++);

                  if((b & 1) == 1)
                        tmp.light();

                  b >>>= 1;
            }

            //    Add and go
            frm.getContentPane().add(myPanel, BorderLayout.CENTER);
            frm.show();

      }

}
```

SerialHandler.java

```
/*
      SerialHandler.java

      by Michael J. Fromberger <sting@linguist.dartmouth.edu>
      Copyright (C) 2000 The Trustees of Dartmouth College

      Handle a simple form of input from the PC serial port, using the Sun
Java
      CommAPI.
 */

import java.io.*;
import java.util.*;
import javax.comm.*;

public class SerialHandler implements SerialPortEventListener
{
      static final int BUFFER_SIZE = 16;

      private InputStream           inStream;
      private SerialPort            commPort;
      private     boolean                 hasData;
      private byte[]                data;
      private int                   numBytes;

      public SerialHandler(String device)
      {
            Enumeration             portList;
            CommPortIdentifier      portID;

            hasData = false;
            data = new byte[BUFFER_SIZE];

            portList = CommPortIdentifier.getPortIdentifiers();
            while(portList.hasMoreElements()) {
                  portID = (CommPortIdentifier)portList.nextElement();

                  //    Scan for serial ports
                  if(portID.getPortType() ==
CommPortIdentifier.PORT_SERIAL) {
                        //    Look for the port we're told to identify
                        if(portID.getName().equals(device)) {
                              System.out.println("Found port: " + device);
                              setupPort(portID);
                        }
                  }
            }
      }     //    end SerialHandler() constructor

      public boolean    hasData() { return hasData; }
```

```
	public byte[]	getData()
	{
		byte[] out;

		synchronized(data) {
			out = new byte[numBytes];

			for(int ix = 0; ix < numBytes; ix++)
				out[ix] = data[ix];

			hasData = false;
			numBytes = 0;
		}

		return out;
	}
	public void	flushData()
	{
		synchronized(data) {
			hasData = false;
			numBytes = 0;
		}
	}

	private void setupPort(CommPortIdentifier portID)
	{
		//	Try to open the serial port itself
		try {
			commPort = (SerialPort)portID.open("SerialHandler",
2000);

		} catch(PortInUseException e) {
			//	ignore this, for now...sigh...
		}

		//	Try to retrieve the input stream to associated with the
port
		try {
			inStream = commPort.getInputStream();

		} catch(IOException e) {
			//	ignore this...
		}

		//	Try to add a new event listener for this port
		try {
			commPort.addEventListener(this);

		} catch(TooManyListenersException e) {
			//	ignore this...
		}

		commPort.notifyOnDataAvailable(true);

		//	We'll hard-wire the parameters, for this application
		try {
```

```
                  commPort.setSerialPortParams(9600,        //    bits per
second
                                    SerialPort.DATABITS_8,        //    8
data bits
                                    SerialPort.STOPBITS_1,        //    1
stop bit
                                    SerialPort.PARITY_NONE);      //    no
parity

            } catch(UnsupportedCommOperationException e) {
                  //    Ignore this, for now...sigh...
            }

      }     //    end of setupPort()

      //    Handle reads from the serial port
      public void serialEvent(SerialPortEvent evt)
      {
            switch(evt.getEventType()) {
                  case SerialPortEvent.DATA_AVAILABLE:

                        try {
                              synchronized(data) {
                                    while(inStream.available() > 0) {
                                          numBytes = inStream.read(data);
                                          /*
                                          System.out.println("data read: "
+ numBytes + " ");
                                          for(int ix = 0; ix < numBytes;
ix++)
                                                System.out.print(data[ix]
+ " ");
                                          System.out.println();
                                           */
                                    }
                                    hasData = true;
                              }
                        } catch(IOException e) {
                              //    Ignore this...
                        }
                        break;

                  default:
                        //    ignore other types of serial events; the
keyboard only sends
                        //    data, anyway...
                        break;
            }

      }     //    end of serialEvent()

}     //    end of class SerialHandler
```

TIME-SORTED HEIRARCHY OF CHORDS

This array represents the averaged data obtained from 10 trials by 10 different people running the **fingers.java** application via the chording keyboard. The data was averaged and sorted using **SortChord.m**.

FingerChord	Completed(0N-1Y)	Time(msec)	#Errors
000100	1.000	1442.182	0.000
000001	1.000	1449.455	0.000
000010	1.000	1608.000	0.000
101000	1.000	1616.000	0.000
001000	1.000	1743.455	0.000
001100	0.909	1868.182	0.545
011000	1.000	1967.455	0.000
000110	1.000	2115.545	0.091
000011	1.000	2116.727	0.182
001010	0.909	2205.909	0.273
010000	1.000	2398.091	0.000
010100	1.000	2403.364	0.091
100100	1.000	2418.909	0.000
010101	1.000	2632.000	0.182
010001	1.000	2645.636	0.182
110000	1.000	2683.727	0.818
010010	1.000	2687.455	0.182
000101	1.000	2867.545	0.091
011100	1.000	2933.273	0.636
100010	1.000	2986.000	0.364
100110	1.000	3042.545	0.636
100011	1.000	3194.636	0.545
110001	1.000	3426.818	0.545
000111	1.000	3449.364	0.273
101010	1.000	3475.727	0.545
100000	1.000	3482.364	0.000
010011	1.000	3547.818	0.636
011010	1.000	3795.364	1.091
010111	0.909	3958.455	0.636
001101	0.909	4320.818	0.455
101001	1.000	4533.909	0.364
001001	1.000	4580.091	0.455
010110	1.000	4698.000	1.000
110100	1.000	5026.364	1.182
100101	1.000	5873.727	0.909
101011	1.000	6792.636	2.636
011001	0.818	6918.182	1.273
001111	0.909	7008.455	2.091
011011	1.000	7291.364	2.818
100111	1.000	7341.182	3.273
001011	0.909	8160.818	2.364
111100	0.909	8324.727	3.182

CHORD-HEIRARCHY (CONTINUED)

FingerChord	Completed(0N-1Y)	Time(msec)	#Errors
101100	0.909	8344.636	2.636
100001	1.000	8432.182	0.091
011101	1.000	9744.818	3.455
011110	0.909	10135.364	5.727
111010	0.818	10340.455	4.273
110010	0.727	10469.455	4.636
111000	0.818	11565.818	5.909
001110	0.909	12266.091	4.000
101101	0.727	12942.273	8.000
110111	0.636	13756.909	12.000
110110	0.727	14586.455	6.182
011111	0.727	14962.455	13.182
111110	0.727	15116.091	8.909
111101	0.727	15508.727	8.273
110011	0.818	15679.727	11.364
111111	0.364	17176.364	9.545
110101	0.818	17727.273	13.818
111011	0.636	18404.545	12.273
101111	0.455	19564.364	10.091
101110	0.455	21971.636	9.545
111001	0.727	23201.455	21.091

"Control" Chord Map Data Files

Started with 6-bit finger chords,

%control.txt: <six bit finger chord value> <control sequence>

```
000011 <ESC>
010111 <CTL-ALT-DEL>
100110 <CTRL-C>
100100 <CTRL-N>
010000 <CTRL-O>
010011 <CTRL-P>
100111 <CTRL-Q>
010001 <CTRL-S>
010110 <CTRL-V>
000111 <CTRL-W>
001111 <CTRL-X>
101000 <SHFT-TAB>
000001 <RETURN>
000010 <SPACE>
000100 <DELETE>
001000 <SHFT-LEFT>
001100 <SHFT-RIGHT>
011000 <SHFT-UP>
000110 <SHFT-DOWN>
```

Converted finger chords to decimal using **DecCord.m** so that they could be piped into perl script-- **ChordPicGenerator**-- generating post script pictures of hands pressing the appropriate chords.

%control3.txt: <decimal chord value> <control sequence>

```
20      <ALT>
 3      <ESC>
25      <"and">
23      <CTL-ALT-DEL>
38      <CTRL-C>
36      <CTRL-N>
16      <CTRL-O>
19      <CTRL-P>
39      <CTRL-Q>
17      <CTRL-S>
22      <CTRL-V>
 7      <CTRL-W>
15      <CTRL-X>
40      <SHFT-TAB>
 1      <RETURN>
 2      <SPACE>
 4      <DELETE>
 8      <SHFT-LEFT>
12      <SHFT-RIGHT>
24      <SHFT-UP>
 6      <SHFT-DOWN>
```

Extended finger chords to 8 bits (the two additional bits determine the "Control" case). This was done via **FullBin.m.**

```
%control2.txt  :  <hex byte chord value>  <control sequence>
C3      <ESC>
D7      <CTL-ALT-DEL>
E6      <CTRL-C>
E4      <CTRL-N>
D0      <CTRL-O>
D3      <CTRL-P>
E7      <CTRL-Q>
D1      <CTRL-S>
D6      <CTRL-V>
C7      <CTRL-W>
CF      <CTRL-X>
E8      <SHFT-TAB>
C1      <RETURN>
C2      <SPACE>
C4      <DELETE>
C8      <SHFT-LEFT>
CC      <SHFT-RIGHT>
D8      <SHFT-UP>
C6      <SHFT-DOWN>
```

"Lower Case" Chord Map Data Files

Started with 6-bit finger chords,

%lower.txt: <six bit binary finger chord value> <character>

```
 010100a
010111 b
100110 c
011100 d
000011 e
100000 f
110001 g
010010 h
010101 i
100101 j
001001 k
000101 l
100011 m
100100 n
010000 o
010011 p
100111 q
110000 r
010001 s
001010 t
100010 u
010110 v
000111 w
001111 x
101010 y
011011 z
001101 -
011010 .
101001 ,
100001 ?
001011 !
110100 :
011101 ;
101011 '
011001 "
101100 (
111100 )
101000 <TAB>
000001 <RETURN>
000010 <SPACE>
000100 <DELETE>
001000 <LEFT>
001100 <RIGHT>
011000 <UP>
000110 <DOWN>
```

Converted finger chords to decimal using **DecCord.m** so that they could be piped into perl script-- **ChordPicGenerator**-- generating post script pictures of hands pressing the appropriate chords.

%lower3.txt: <decimal chord value> <character>

```
20  a
23  b
38  c
28  d
3   e
32  f
49  g
18  h
21  i
37  j
9   k
5   l
35  m
36  n
16  o
19  p
39  q
48  r
17  s
10  t
34  u
22  v
7   w
15  x
42  y
27  z
13  -
26  .
41  ,
33  ?
11  !
52  :
29  ;
25  '
43  "
44  (
60  )
40  <TAB>
1   <RETURN>
2   <SPACE>
4   <DELETE>
8   <LEFT>
12  <RIGHT>
24  <UP>
6   <DOWN>
```

Extended finger chords to 8 bits (the two additional bits determine the "Lower" case). This was done via **FullBin.m.**

%lower2.txt : <hex byte chord value> <character>

```
 14      a
17      b
26      c
1C      d
03      e
20      f
31      g
12      h
15      i
25      j
09      k
05      l
23      m
24      n
10      o
13      p
27      q
30      r
11      s
0A      t
22      u
16      v
07      w
0F      x
2A      y
1B      z
0D      -
1A      .
29      ,
21      ?
0B      !
34      :
1D      ;
2B      '
19      "
2C      (
3C      )
28      <TAB>
01      <RETURN>
02      <SPACE>
04      <DELETE>
08      <LEFT>
0C      <RIGHT>
18      <UP>
06      <DOWN>
```

"Math&Symbol" Chord Map Data Files

Started with 6-bit finger chords.

%mathsym.txt: <six bit binary finger chord value> <character>

```
010100 @
010111 <
100110 _
011100 3
000011 >
100000 1
110001 7
010010 5
010101 |
100101 {
001001 [
000101 4
100011 8
100100 /
010000 O
010011 +
100111 ^
110000 2
010001 $
001010 =
100010 9
010110 ]
000111 *
001111 %
101010 6
011011 #
001101 -
011010 ~
101001 `
001011 \
101011 }
011001 &
101100 (
111100 )
101000 <TAB>
000001 <RETURN>
000010 <SPACE>
000100 <DELETE>
001000 <LEFT>
001100 <RIGHT>
011000 <UP>
000110 <DOWN>
```

Converted finger chords to decimal using **DecCord.m** so that they could be piped into perl script-- **ChordPicGenerator**-- generating post script pictures of hands pressing the appropriate chords.

%mathsym3.txt: <decimal chord value> <character>

```
 20      @
23       <
38       _
28       3
 3       >
32       1
49       7
18       5
21       |
37       {
 9       [
 5       4
35       8
36       /
16       O
19       +
39       ^
48       2
17       $
10       =
34       9
22       ]
 7       *
15       %
42       6
27       #
13       -
26       ~
41       `
11       \
43       }
25       &
44       (
60       )
40       <TAB>
 1       <RETURN>
 2       <SPACE>
 4       <DELETE>
 8       <LEFT>
12       <RIGHT>
24       <UP>
 6       <DOWN>
```

Extended finger chords to 8 bits (the two additional bits determine the "Math&Symbol" case). This was done via **FullBin.m.**

```
%mathsym2.txt  :  <hex byte chord value>  <character>
 94      @
97       <
A6       _
9C       3
83       >
A0       1
B1       7
92       5
95       |
A5       {
89       [
85       4
A3       8
A4       /
90       O
93       +
A7       ^
B0       2
91       $
8A       =
A2       9
96       ]
87       *
8F       %
AA       6
9B       #
8D       -
9A       ~
A9       `
8B       \
AB       }
99       &
AC       (
BC       )
A8       <TAB>
81       <RETURN>
82       <SPACE>
84       <DELETE>
88       <LEFT>
8C       <RIGHT>
98       <UP>
86       <DOWN>
```

"Upper" Chord Map Data Files

Started with 6-bit finger chords.

%upper.txt: <six bit binary finger chord value> <character>

```
 010100A
010111 B
100110 C
011100 D
000011 E
100000 F
110001 G
010010 H
010101 I
100101 J
001001 K
000101 L
100011 M
100100 N
010000 O
010011 P
100111 Q
110000 R
010001 S
001010 T
100010 U
010110 V
000111 W
001111 X
101010 Y
011011 Z
001101 -
011010 .
101001 ","
100001 ?
001011 !
110100 :
011101 ;
101011 '
011001 """
101100 (
111100 )
101000 <TAB>
000001 <RETURN>
000010 <SPACE>
000100 <DELETE>

001000 <LEFT>
001100 <RIGHT>
011000 <UP>
000110 <DOWN>
```

Converted finger chords to decimal using **DecCord.m** so that they could be piped into perl script-- **ChordPicGenerator**-- generating post script pictures of hands pressing the appropriate chords.

%upper3.txt: <decimal chord value> <character>

```
 20    A
23     B
38     C
28     D
 3     E
32     F
49     G
18     H
21     I
37     J
 9     K
 5     L
35     M
36     N
16     O
19     P
39     Q
48     R
17     S
10     T
34     U
22     V
 7     W
15     X
42     Y
27     Z
13     -
26     .
41     ,
33     ?
11     !
52     :
29     ;
25     '
43     "
44     (
60     )
40     <TAB>
 1     <RETURN>
 2     <SPACE>
 4     <DELETE>
 8     <LEFT>
12     <RIGHT>
24     <UP>
 6     <DOWN>
```

Extended finger chords to 8 bits (the two additional bits determine the "Upper" case). This was done via **FullBin.m.**

%upper2.txt : <hex byte chord value> <character>

```
54      A
57      B
66      C
5C      D
43      E
60      F
71      G
52      H
55      I
65      J
49      K
45      L
63      M
64      N
50      O
53      P
67      Q
70      R
51      S
4A      T
62      U
56      V
47      W
4F      X
6A      Y
5B      Z
4D      -
5A      .
69      ,
61      ?
4B      !
74      :
5D      ;
6B      '
59      "
6C      (
7C      )
68      <TAB>
41      <RETURN>
42      <SPACE>
44      <DELETE>
48      <LEFT>
4C      <RIGHT>
58      <UP>
46      <DOWN>
```

CODE FOR CREATING A HEIRARCHY OF CHORDS, FROM "MOST INTUITIVE" TO "LEAST INTUITIVE"

This program reads in a number of chord data files produced by **Fingers.java**, sorts them by chord, averages their values to create an "average" array, then sorts this average array by time and prints the "average" chord data to file. Finally, the program plots chord rank vs time, completion, and errors.

SortChord.m

```
%SortChord.m

%This program reads in a number of chord data files,
%sorts them by chord, averages their values to create
%an "average" array, then sorts this average array by time
%and prints the "average" chord data to file.
%Finally, the program plots chord rank vs time, completion, and
%errors.



%The array of input file names
in_names = {'KATE.TXT', 'KRISPIN.TXT', 'LAURENN.TXT', ...
       'LEAHANN.TXT', 'MITCHY.TXT', ...
       'SHOM.TXT', 'STEVER.TXT', 'WILLN.TXT', ...
      'joe.txt', 'scottmilne.txt', 'susan.txt'};



%One by one, read the files into arrays, and do stuff
for i=1:11

    %Read in file
    in_path = strcat('ResultsV1/', in_names{i});
    fid = fopen(in_path);
    [hex_chord, done, time, errors] = ...
      textread(in_path, '%s %d %d %d', 'commentstyle','matlab');
    fclose(fid);


    %Convert hexidecimal chord values to decimal
    dec_chord = hex2dec(hex_chord);


    %Place info in an array of 2D arrays, and sort based on chord decimal
    %values
```

```
    indata_array(:,:,i) = [dec_chord, done, time, errors];
    indata_array(:,:,i) = sortrows(indata_array(:,:,i));

end;


%Calculate one "average" array
for c=1:4
    for r=1:63
      ave_array(r,c) = mean(indata_array(r,c,:));
    end;
end;


%Sort average array based on times
format short
ave_array = sortrows(ave_array, [3]);
disp(ave_array)




%Save average array to a text file
fid = fopen('ResultsV2/ave_array.txt', 'w');

fprintf(fid, 'Mean Chord Data Values\n');
fprintf(fid, ['FingerChord      Done(0-1)        Time(msec)' ...
            '          Errors\n']);

for i=1:63

    %Convert chords to binary, 6 digits
    chord = num2str(dec2bin(ave_array(i,1)));
    numleadzeros = 6-length(chord);
    leadzeros = '';
    for z=1:numleadzeros
      leadzeros = strcat(leadzeros,'0');
    end;
    chord = strcat(leadzeros, chord);


    %Print data to file
    fprintf(fid,  '%s           %.3f            %.3f            %.3f\n',
...
          chord, ave_array(i,2), ave_array(i,3), ave_array(i,4)  );
end;
fclose(fid);




%Graph Chord Heirarchy vs. Chord Completion Time
i = 1:63;
grid on;
subplot(3,1,1)
plot(i, ave_array(i,3), '+-');
```

```
title('Heuristics for FingerChord Intuitiveness -- Based on Averaged
Data');
xlabel('Chord Rank in Heirarchy (1:Most Intuitive, 63:Least Intuitive)');
ylabel('Ave.CompletionTime(msec)');
legend('Ave. Successful Chord Completion Time (msec)',0);

%Graph Chord Heirarchy vs. Successful Chord Completion Value
subplot(3,1,2)
plot(i, ave_array(i,2), '*-');
xlabel('Chord Rank in Heirarchy (1:Most Intuitive, 63:Least Intuitive)');
ylabel('Ave.OfSuccessfulCompletionBinaryValue');
legend('Average of Successful Chord Completion Binary Value (1=completed,
0=skipped)',0);

%Graph Chord Heirarchy vs. Number of Errors
subplot(3,1,3)
plot(i, ave_array(i,4), 's:');
xlabel('Chord Rank in Heirarchy (1:Most Intuitive,  63:Least Intuitive)');
ylabel('NumberOfErrors');
legend('Number of Errors Before a Successful Completion or Skip',0);
```

CODE FOR EXTENDING 6-BIT FINGER BUTTON CODES TO 8-BIT CHORD-MAP CODES

This MATLAB function reads in files holding arrays in the following
form:

```
[6 bit binary finger chord]  [character the finger chord maps to]
[6 bit binary finger chord]  [character the finger chord maps to]
           .                              .
           .                              .
```

Based on the file's 'case' (upper, lower, mathsym or control), the program extends the six bit binary chord value to an 8 bit binary value, translates it to hex and writes the modified file.

It requires an argument (1 2 3 or 4) corresponding to the case of the data file it will read in: upper, lower, mathsym or control.

FullBin.m

```
%FullBin.m
function FullBin(i)

%This function reads in files holding arrays in the following
%form:
%
%[6 bit binary finger chord]  [character the finger chord maps to]
%
%Based on the file's 'case' (upper, lower, mathsym or control),
%The program extends the six bit binary chord value to an 8 bit binary
%value, translates it to hex and writes the modified file.



%The array of input & output file names
in_names = {'lower.txt' 'upper.txt' 'mathsym.txt' 'control.txt'};
out_names = {'lower2.txt' 'upper2.txt' 'mathsym2.txt' 'control2.txt'};
```

```
%Read file into array, and do stuff

%Read in file
fid = fopen(in_names{i});
[chords characters] = ...
    textread(in_names{i}, '%s %s');
fclose(fid);



%Extend chords to 8 bits
if i == 1
    chords = strcat('00', chords);
elseif i== 2
    chords = strcat('01', chords);
elseif i == 3
    chords = strcat('10', chords);
else % i == 4
    chords = strcat('11', chords);
end;



%Convert chords to hex strings
dec_chords = bin2dec(chords);
hex_chords = num2str(dec2hex(dec_chords));



%Convert hex_chords from cell array to array
for v=1:length(hex_chords)
    temp{v} = hex_chords(v,:);
end;
hex_chords = temp';



%Place info in an array of 2D arrays
data_array = [hex_chords, characters];



%Save average array to a text file
fid = fopen(out_names{i}, 'w');
for r=1:length(data_array)
    %Print data to file
    chordval = data_array(r,1);
    charval = data_array(r,2);
    fprintf(fid,  '%s          %s\n', ...
          chordval{:}, charval{:}   );
end;
fclose(fid);
```

CODE FOR CONVERTING 6-BIT FINGER CODES TO THEIR DECIMAL EQUIVALENTS WITHIN [FINGER-CODE CHARACTER] ARRAYS

This MATLAB function reads in files holding arrays in the following
form:

[6 bit binary finger chord] [character the finger chord maps to]
[6 bit binary finger chord] [character the finger chord maps to]
. .
. .

Based on the file's 'case' (upper, lower, mathsym or control), the program translates the 6 bit binary finger code into decimal, then writes to a file.

It requires an argument (1 2 3 or 4) corresponding to the case of the file it will read in: upper, lower, mathsym or control.

DecCord.m

```
%DecChord.m
function DecChord(i)

%This function reads in files holding arrays in the following
%form:
%
%[6 bit binary finger chord]  [character the finger chord maps to]
%
%Based on the file's 'case' (upper, lower, mathsym or control),
%The program translates the 6 bit binary finger code into decimal,
%then writes the modified file.


%The array of input & output file names
in_names = {'lower.txt' 'upper.txt' 'mathsym.txt' 'control.txt'};
out_names = {'lower3.txt' 'upper3.txt' 'mathsym3.txt' 'control3.txt'};

%Read file into array, and do stuff

%Read in file
fid = fopen(in_names{i});
```

```
[chords characters] = ...
    textread(in_names{i}, '%s %s');
fclose(fid);



%Convert chords to decimal number strings
dec_chords = num2str(bin2dec(chords));



%Convert hex_chords from cell array to array
for v=1:length(dec_chords)
    temp{v} = dec_chords(v,:);
end;
dec_chords = temp';



%Place info in a 2D array
data_array = [dec_chords, characters];



%Save average array to a text file
fid = fopen(out_names{i}, 'w');
for r=1:length(data_array)
    %Print data to file
    chordval = data_array(r,1);
    charval = data_array(r,2);
    fprintf(fid,  '%s           %s\n', ...
          chordval{:}, charval{:}   );
end;
fclose(fid);
```

HAND-CHORD PICTURE GENERATION CODE

Written by Michael Fromberger

ChordPicGenerator

```
#!/usr/bin/perl

#
# Generate hand images in EPS for the chording keyboard
#
# by Michael J. Fromberger <sting@linguist.dartmouth.edu>
# Copyright (C) 2000 The Trustees of Dartmouth College
#
# $Id: handy,v 1.2 2000/05/25 13:44:01 sting Exp $
#
@_=split(/\//,$0);chomp($prog=pop(@_));

# These numbers came from the original static hand image we used for
# the reaction tester (which, in turn, came from an on-screen tracing
# of my left hand, as rendered into an outline using Canvas 6 on the
# Macintosh, and exported as a PICT.  Needless to say, these are ad
# hoc values, but that's what I based the outlines below on...
$BASEWIDTH = 331;
$BASEHEIGHT = 378;
$SCALE = 0.15; # should be a percentage (used as a multiplier)

%lower = (
        "1" => '<RETURN>', "2" => '<SPACE>',    "3" => 'e',
        "4" => '<DELETE>', "5" => 'l',          "6" => '<DOWN>',
        "7" => 'w',        "8" => '<LEFT>',     "9" => 'k',
        "10" => 't',       "11" => '!',         "12" => '<RIGHT>',
        "13" => '-',       "15" => 'x',         "16" => 'o',
        "17" => 's',       "18" => 'h',         "19" => 'p',
        "20" => 'a',       "21" => 'i',         "22" => 'v',
        "23" => 'b',       "24" => '<UP>',            "25" => '\'',
        "26" => '.',       "27" => 'z',         "28" => 'd',
        "29" => ';',       "32" => 'f',         "33" => '?',
        "34" => 'u',       "35" => 'm',         "36" => 'n',
        "37" => 'j',       "38" => 'c',         "39" => 'q',
        "40" => '<TAB>',   "41" => ',',         "42" => 'y',
        "43" => '"',       "44" => '(',         "48" => 'r',
        "49" => 'g',       "52" => ':',         "60" => ')',
);
%upper = (
        "1" => '<RETURN>', "2" => '<SPACE>',     "3" => 'E',
```

```
          "4" => '<DELETE>',  "5" => 'L',            "6" => '<DOWN>',
          "7" => 'W',         "8" => '<LEFT>',       "9" => 'K',
          "10" => 'T',        "11" => '!',           "12" => '<RIGHT>',
          "13" => '-',        "15" => 'X',           "16" => 'O',
          "17" => 'S',        "18" => 'H',           "19" => 'P',
          "20" => 'A',        "21" => 'I',           "22" => 'V',
          "23" => 'B',        "24" => '<UP>',              "25" => '\'',
          "26" => '.',        "27" => 'Z',           "28" => 'D',
          "29" => ';',        "32" => 'F',           "33" => '?',
          "34" => 'U',        "35" => 'M',           "36" => 'N',
          "37" => 'J',        "38" => 'C',           "39" => 'Q',
          "40" => '<TAB>',    "41" => ',',           "42" => 'Y',
          "43" => '"',        "44" => '(',           "48" => 'R',
          "49" => 'G',        "52" => ':',           "60" => ')',
);
%control = (
        "1" => '<RETURN>',   "2" => '<SPACE>',     "3" => '<ESC>',
         "4" => '<DELETE>',  "6" => '<SHFT-DOWN>',       "7" => '<CTRL-W>',
         "8" => '<SHFT-LEFT>',       "12" => '<SHFT-RIGHT>',      "15" => '<CTRL-X>',
         "16" => '<CTRL-O>', "17" => '<CTRL-S>',  "19" => '<CTRL-P>',
         "20" => '<ALT>',    "22" => '<CTRL-V>',  "23" => '<CTL-ALT-DEL>',
         "24" => '<SHFT-UP>',        "25" => '<"and">',    "36" => '<CTRL-N>',
         "38" => '<CTRL-C>', "39" => '<CTRL-Q>',  "40" => '<SHFT-TAB>',
);
%math = (
        "1" => '<RETURN>',   "2" => '<SPACE>',     "3" => '>',
        "4" => '<DELETE>',   "5" => '4',           "6" => '<DOWN>',
        "7" => '*',          "8" => '<LEFT>',      "9" => '[',
        "10" => '=',         "11" => '\\',         "12" => '<RIGHT>',
        "13" => '-',         "15" => '%',          "16" => 'O',
        "17" => '$',         "18" => '5',          "19" => '+',
        "20" => '@',         "21" => '|',          "22" => ']',
        "23" => '<',         "24" => '<UP>',              "25" => '&',
        "26" => '~',         "27" => '#',          "28" => '3',
        "32" => '1',         "34" => '9',          "35" => '8',
        "36" => '/',         "37" => '{',          "38" => '_',
        "39" => '^',         "40" => '<TAB>',      "41" => '`',
        "42" => '6',         "43" => '}',          "44" => '(',
        "48" => '2',         "49" => '7',          "60" => ')',
);

%kind = (
          "math" => \%math,
          "lower" => \%lower,
          "upper" => \%upper,
          "control" => \%control
);
```

```
foreach $k (keys %kind) {
   print STDERR $k, " ";
   $v = $kind{$k};

   for($ix = 0; $ix < 64; $ix++) {
        next unless exists($v->{$ix}); # skip undefined chords

        $fn = sprintf("%s%02d.eps", $k, $ix);
        open(OFP, ">$fn") or die "$prog: can't open $fn: $!\n";

        # Choose a size for the label based on the length of it
        $label = $v->{$ix};
        if(length($label) == 1) {
           $size = 200;
        } elsif(length($label) < 8) {
           $size = 50;
        } elsif(length($label) < 12) {
           $size = 40;
        } else {
           $size = 30;
        }

        print_ps(OFP, $k, $ix, $size, $label, $SCALE);

        close(OFP);
        print STDERR ".";
   }
   print STDERR "\n";
}
exit 0;

#--------------------------------------------------------------------

sub print_ps {
   my($ofp, $kind, $value, $basefont, $string, $scale) = @_;
   my($width, $height, $op);

   $width = int($BASEWIDTH * $scale) + 1;
   $height = int($BASEHEIGHT * $scale) + 1;
   $string =~ s/([()])/\\\1/g;

   print $ofp <<EOHEADER;
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 $width $height
%%Title: ($kind $value $string)
%%IncludeFont: Times-Roman

/pt { $scale mul } def  % point scaling
```

```
EOHEADER

   for (1..6) {
        if($value & 1) {
           $op = "fill";
        } else {
           $op = "stroke";
        }

        printf $ofp ("/op%d { %s } def\n", $_, $op);
        $value >>= 1;
   }

   printf $ofp ("\n/ch (%s) def\n/fs %d pt def\n\n",
                $string, $basefont);

   print $ofp <<EOBODY;
/lw 0.5 def          % line weight
/cf 0 def            % fill shading
/cr 11 pt def        % circle radius
/ff { /Times-Roman findfont fs scalefont setfont } def

newpath
% Initial point
318 pt 2 pt moveto

% Bottom curve of thumb
230 pt 4 pt 160 pt 17 pt 380 pt arct
160 pt 17 pt 57 pt 72 pt 380 pt arct
57 pt 72 pt lineto

% Tip and inner curve of thumb
17 pt 93 pt 31 pt 125 pt 85 pt 115 pt curveto
123 pt 92 pt lineto
149 pt 89 pt 155 pt 117 pt 121 pt 156 pt curveto

% Outer line of index finger
23 pt 240 pt lineto

% Tip of index finger
0 pt 266 pt 0 pt 300 pt 52 pt 270 pt curveto

% Inner line of index finger
120 pt 217 pt lineto

% Joint and outer line of middle finger
138 pt 195 pt 149 pt 208 pt 118 pt 241 pt curveto
64 pt 298 pt lineto
```

```
% Tip of middle finger
29 pt 343 pt 57 pt 364 pt 92 pt 329 pt curveto
152 pt 260 pt lineto

% Joint and outer line of ring finger
177 pt 233 pt 188 pt 244 pt 161 pt 278 pt curveto
115 pt 343 pt lineto

% Tip of ring finger
101 pt 368 pt 142 pt 378 pt 158 pt 347 pt curveto
209 pt 269 pt lineto

% Joint and outer line of pinky
228 pt 242 pt 237 pt 251 pt 224 pt 277 pt curveto
194 pt 325 pt lineto

% Tip of pinky
179 pt 354 pt 206 pt 367 pt 231 pt 329 pt curveto
271 pt 263 pt lineto

% Outside curve of hand
307 pt 215 pt lineto
331 pt 182 pt lineto

0 setgray lw setlinewidth stroke

% Circles
newpath 208 pt cr add 333 pt moveto 208 pt 333 pt cr 0 360 arc
cf setgray lw setlinewidth op1
newpath 132 pt cr add 350 pt moveto 132 pt 350 pt cr 0 360 arc
cf setgray lw setlinewidth op2
newpath 90 pt cr add 300 pt moveto 90 pt 300 pt cr 0 360 arc
cf setgray lw setlinewidth op3
newpath 65 pt cr add 328 pt moveto 65 pt 328 pt cr 0 360 arc
cf setgray lw setlinewidth op4
newpath 53 pt cr add 243 pt moveto 53 pt 243 pt cr 0 360 arc
cf setgray lw setlinewidth op5
newpath 26 pt cr add 266 pt moveto 26 pt 266 pt cr 0 360 arc
cf setgray lw setlinewidth op6

% Set the current font and go to the base location
ff newpath 220 pt 65 pt moveto

% Get the string's x displacement, cut it in half, and back up
% by that amount, so the string is centered over the target point
ch stringwidth pop 2 div neg 0 rmoveto
ch show
```

```
showpage
EOBODY

}

#-------------------------------------------------------------
# HERE THERE BE DRAGONS
```

CALCULATOR LEARNABILITY TEST CODE

The following MATLAB program was written to test the learnability of a reduced character set: numbers and math operators. It times data entry sequences, and saves times to a file.

Calc.m

```
%Calc.m
function calc

%A program testing a limited character set consisting of
%numbers and the operands: [+,-,*,/]. Basically its a timed
%calculator.
%Trial time results get saved in a file



%Variables & Constants
L = 20;    % 2/3 the number of questions
j=1;       % the current question number



%Get User Name
name_str = input(['  Please enter your first name in lowercase'...
             ' letters, \nthen press <return>: '], 's');
file_name_str = [name_str '.mat'];



%Directions
disp(' ');
disp('  The following program is designed to test how fast you can');
disp(' accurately enter numbers and math operators (+,-,* and /).');
disp('  At each of the following (30) prompts, please repeat-type the');
disp('characters enclosed in <<>> brackets, then press <return>.');
disp(' ');
disp('  For example:');
disp('1.  <<42>>42');
disp('2.  <<*>>*');
disp('and so on...');
disp(' ');
input('  Press <return> to begin:');



%start timer
tic;
```

```
%Create an array of L random integer numbers between 1 and 100, no zeros
%rand_numbers = random('Discrete Uniform', 100*ones(1,L));
rand_numbers = floor(100*rand([1 L]))+1;
while (~isempty(find(rand_numbers == 0)))
    rand_numbers = floor(100*rand([1 L]))+1;
end;



%Create an array of operands:  ['+','-', '*' '/'] ,and an array of
%random indeces for this operand array
operand_symbols = ['+', '-', '*' '/'];
rand_operand_codes = floor(4*rand([1 L]))+1;
while (~isempty(find(rand_numbers == 0)))
    rand_operand_codes = floor(4*rand([1 L]))+1;
end;




%Main program loop
for i = 1:2:length(rand_numbers)-1

    %Get (and error check) first operand
    op1 = ' ';
    first_time = 1;
    while ~isa(op1, 'double') | op1 ~= rand_numbers(i)

      if first_time ~= 1
          disp(['Mistake made. Please type <<'num2str(rand_numbers(i)) ...
              '>>.']);
      else
          disp(' ');
      end;

      op1 = input([num2str(j) '. <<' num2str(rand_numbers(i)) ...
                '>>'], 's');
      op1 = str2num(op1);
      first_time = 0;
    end;
    j=j+1;



    %Get (and error check) operator
    operator = ' ';
    first_time = 1;
    while ~strcmp(operator,operand_symbols(rand_operand_codes(i)))
      if first_time ~= 1
          disp(['Please type <<' operand_symbols(rand_operand_codes(i))
...
              '>>']);
      else
          disp(' ');
      end;
      operator = input([num2str(j) '. <<' ...
                    operand_symbols(rand_operand_codes(i))...
```

```
                    '>>'], 's');
      first_time = 0;
    end;
    j=j+1;



    %Get (and error check) second operand
    op2 = ' ';
    first_time = 1;
    while ~isa(op2, 'double') | op2 ~= rand_numbers(i+1)

      if first_time ~= 1
          disp(['Please type <<' num2str(rand_numbers(i+1)) '>>.']);
      else
          disp(' ');
      end;
      op2 = input([num2str(j) '. <<' num2str(rand_numbers(i+1)) '>>'],...
                's');
      op2 = str2num(op2);
      first_time = 0;
    end;
    j=j+1;



end; %for



%Stop stopwatch
time_taken = toc


%Save the time taken
 if exist(file_name_str, 'file')
     load(file_name_str);
     times(length(times)+1) = time_taken;
 else
     times(1) = time_taken;
 end;

 save(file_name_str, 'times');


%Thanx
disp(' ');
disp('  Thanks for taking the time to do this trial!');
disp(' ');
disp(['  Your time this time round was: ' num2str(time_taken) ...
     ' seconds.']);
disp(' ');
disp('  Your trial times are:');
disp(' ');
disp(num2str(times));
```

CALCULATOR LEARNABILITY TEST RESULTS CODE

The following MATLAB program displays in graphical format the data obtained using "Calc.m". The graphs are supposed to represent the "learnability" of the interface for a reduced character set.

CalcGraph.m

```
%CalcGraph.m
% Graphs the results of the "Calc.m" Test, which prompted
% test subjects to enter a set number of numbers and math symbols
% and timed each trial.  This program graphs trial number vs time
% taken.


%The results:
ayorkor = [509.1375  353.9590  321.2991];
dudley = [3839.0 378.];
jennyl = [486.3291  193.2480];
kl = [243.089 255.567 206.117 195.121 178.156 189.172 152.059...
      164.586 156.254];
leah = [356.0343  290.2411  456.0875];
michael = [281  218.2940];
zoe = [34882 401];
dean = [12052 395 405];
greg = [33343 12126 468];

%Time with a standard keyboard
%benchmarq = mean([52.5860 46.7869 39.3441])

%Graph Results
subplot(3,3,1)
plot([1:length(ayorkor)], ayorkor, '+-');
xlabel('Trial Number');
ylabel('Trial Time (secs)');

subplot(3,3,2)
plot([1:length(dudley)], dudley, '*-');
xlabel('Trial Number');
ylabel('Trial Time (secs)');

subplot(3,3,3)
plot([1:length(jennyl)], jennyl, 's-');
xlabel('Trial Number');
ylabel('Trial Time (secs)');

subplot(3,3,4)
plot([1:length(leah)], leah, 'o-');
xlabel('Trial Number');
ylabel('Trial Time (secs)');
```

```
subplot(3,3,5)
plot([1:length(zoe)], michael, 'o-');
xlabel('Trial Number');
ylabel('Trial Time (secs)');

subplot(3,3,6)
plot([1:length(dean)], dean, 'o-');
xlabel('Trial Number');
ylabel('Trial Time (secs)');

subplot(3,3,7)
plot([1:length(greg)], greg, 'o-');
xlabel('Trial Number');
ylabel('Trial Time (secs)');

subplot(3,3,8:9)
ten_hours = [1:60];
Y = 230*exp(-.11*ten_hours) + 60;
plot([1:length(kl)], kl, 'o-',  ten_hours, Y, 'k:', 60,Y(60),'k*');
xlabel('Trial Number');
ylabel('Trial Time (secs)');
legend('data', 'fitted curve', 'time after 10hrs (extrapolated)',0);
hold off



diary 'CalcRes';

%Speed prediction after 10 of testing hours:

%Assumptions:
%1) Ten minutes necessary per trial => 60 trials can be done in
% 10 hours.

%2) Each trial, hit 90 characters

%3) Five characters in a word (average)
%http://ds.dial.pipex.com/town/park/yfn77/Academic/Maths/Sentence.html

final_trial_time = Y(60);
chars_per_sec=90/final_trial_time;
chars_per_min=chars_per_sec*60;
words_per_min=chars_per_min/5;
disp(['Predicted words per min after 10 hours of testing (60 trials):' ...
      num2str(words_per_min)]);
diary off;
```

Final Prototype Survey

1) What's good about this interface? What are its strengths?

2) What are the interface's weaknesses?

3) Does anything in particular bother you about using the device?

4) Could you imagine using this device for typing? Could you imagine using it for some other purpose?

5) If you can envision using this device, in what setting do you envision using it?

6) If you were to use this device for something other than entering text into a a digital device (computer, digital assistant, etc), what would you use it for?

7) What would you call this device?

8)Please rate the device's acceptability with a number 1(low) through 5(high), taking into account aesthetic, social, psychological & physiological considerations, such as:
Is it ugly?
Would I feel comfortable using this device in public?
Is using the device comfortable? Does it hurt to use it? (Assuming it is properly adjusted and you have been using it for a few weeks already.)

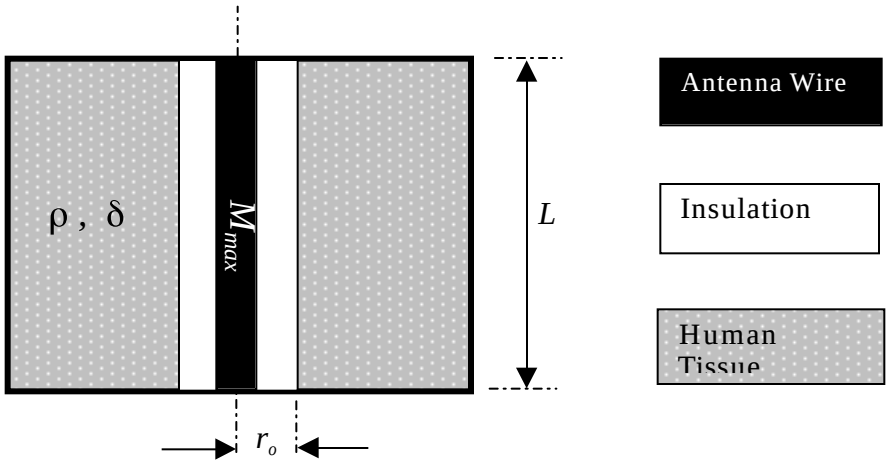
HID Project	Krispin Leydon								
FINAL PROTOTYPE SURVEY RESULTS									
Name	**Profession**	**Interface PROS**	**Interface Cons**	**Could Imagine Typing with device**	**Envisioned useful settings**	**Alternate functions**	**Programmable Chord Map Desirable?**	**Acceptability Rating**	
Test Subject 1	ES Student	one-handed, comfortable,naturat	Some chords awkward, practice required	Yes	Web browsing, taking notes, quick e-mails	remote control		4	
Test Subject 2	ES Student	natural, fun, good weight, adjustable	learning time, need better tactile feedbak, better mode shifting, padded grip region, better buttons	Yes		remote control, virtual reality aplications		4	
Test Subject 3	ES Student	one-handed, mobility	some awkward button combos, device slipped from grip	Yes	Handycapped access, extensor robotics	video game controller, all purpose controller		5	
Test Subject 4	ES Student	mobilty	Buttons don't provide enough feedback, grip needs cushioning	Yes	while travelling, an idea journal	all purpose controler	yeah!	4.5	
Test Subject 5	ES Student	light, comfortable, fast to learn	Would take practice to get fast	Yes				4	
Test Subject 6	ES Student	don't have to sit down, adjustable	Too big, doens't fit hand, have to learn chords, wires exposed	Yes	astronaut, typing on a hike		yes!	4	
Test Subject 7	Woodshop Instructor	One-handed operation.	Difficult to learn, some chords awkward, cramped hands	Yes		PDAs, remotes for vehicles, tv		4	
Test Subject 8	Woodshop Instructor	Free range of motion, good size & weight	Frustrating to learn, some chords impossible	Yes	remote, PDA, all-purpose controller			3.3	
		natural, one-handed, mobility, light comfortable, don' have to sit, free range of motion, good size&weight	**awkward chords, difficult to learn, buttons didn't provide enough tactile feedback, wires exposed, some people experienced hand cramps, grip needs cushioning.**	Yes(100%)	**remote,video games, PDAs, extensor robotics, web navigation, handicaped access**		Y(100%)	**ave: 4.1**	

Analysis:**Risk Posed by Head-Mounted RF Transmitter:**
A Worst Case Analysis:

FCC Safety Spec: Mass-normalized rate of energy absorption (SAR) is limited to 1.6 W/Kg averaged over any 1g of tissue[1].

Assumptions:
•Antenna is an insulated wire of length L with an insulation thickness of r_0.
•Wire is surrounded on all sides by human tissue.
•Radio is constantly transmitting all of the power it receives: M_{max}.
•Radio energy is emitted with even intensity along the antennas length.
•The penetration depth for power in human tissue is a constant δ .
•Human tissue is a constant density ρ

Idealized Model:



[1] Adam D. Tinniswood, et al., "Computations of SAR Distributions for Two Anatomically Based Models of the Human Head Using CAD Files of Commercial Telephones and the Parallelized FDTD Code", IEEE Transactions On Antennas and Propagation, Vol. 46, No. 6 June 1998. p829.

Analysis:

Given our assumptions and model, the maximum power density released by our antenna into a gram of tissue is :

$$M_{Max} = \frac{k2\pi L}{\rho}\int_{r_o}^{\infty} e^{-\frac{r}{\delta}} dr \qquad \frac{W}{Kg} \tag{1}$$

where *k* is an unknown constant of proportionality.

To isolate *k*, we solve the integral

$$M_{Max} = \frac{K2\pi L}{\rho}\left[1 - e^{-\frac{r_0}{\delta}}\right] \qquad \frac{W}{Kg} \tag{2}$$

and rearrange the terms:

$$k = \frac{\delta\rho}{2\pi L} M_{Max} e^{\frac{r_0}{\delta}} \qquad \frac{Wm^3}{Kg^2} \tag{3}$$

To find a value for k, we enter the following values in equation (2):

M_{max} = 5V•1.5mA = 7.5mW
δ = .03m.
L = 0.1702m
ρ = 1000kg/m^3

And obtain

$$k = .2035\frac{Wm^3}{Kg^2}$$

Entering this value for *k* into equation (2), we find that the maximum power density released by our antenna into a gram of human tissue is

$$.0073\frac{W}{Kg} << 1.6\frac{W}{Kg}.$$

Conclusion:

Since the power density released by our antenna is several orders of magnitude below the FCC limit, we will assume that the radio emission from our device is not a significant health risk.

HUMAN-SIDE MATRIX 1: Human Control Method
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
Hand/Finger Motion	**4**	**4**	**5**	**5**	**3**	*	*	*	*	**97**
Voice	4	5	3	4	2	*	*	*	*	88
Eye Motion	4	4	2	3	2	*	*	*	*	70
Body Motion	4	5	2	3	3	*	*	*	*	79
Nerve Impulses	5	4	3	3	1	*	*	*	*	76

HUMAN-SIDE MATRIX 2: Physical Interface Topology

1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
Wearable	5	4	2	*	4	5	3	*	3	109
No Physical Form	5	5	3	*	5	2	2	*	1	82
(2)Handheld	3	4	3	*	2	5	3	*	4	103
(1)Handheld	**4**	**3**	**5**	*	**4**	**5**	**3**	*	**5**	**123**

HUMAN-SIDE MATRIX 3: Actuation Type
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
Linear Motion	3	4	5	3	3	3	2	*	3	113
Linear Force	4	3	3	2	3	3	2	*	4	111
Discrete Motion	5	4	5	4	4	5	5	*	4	155
Discrete Force	5	3	3	4	4	5	3	*	4	140
Discrete Motion&Force	**5**	**4**	**5**	**5**	**5**	**5**	**5**	*	**5**	**165**

HUMAN-SIDE MATRIX 4: DISCRETE CONTROL SCHEME
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
Serial Code	5	2	1	1	4	5	3	4	*	100
Rotary Selection	5	3	3	3	2	3	4	4	*	118
One Actuator per Char.	2	4	4	2	5	3	5	2	*	110
Chording	4	5	3	4	4	5	4	5	*	132
Rotary-Chording Combo	**5**	**5**	**3**	**5**	**5**	**4**	**3**	**5**	*	**152**

HUMAN-SIDE MATRIX 5: ONE-SIZE-FITS-ALL OR ADJUSTABLE?

1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
One-Size-Fits-All	*	3	5	3	*	5	5	3	5	121
Adjustable	*****	**5**	**4**	**5**	*****	**4**	**3**	**5**	**3**	**123**

DEVICE-SIDE MATRIX 1: Digital Design Approach
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
VLSI	5	*	*	*	*	3	3	1	2	59
State Machine & Glue Logic	3	*	*	*	*	3	4	2	3	58
Microcontroller	**4**	*	*	*	*	**5**	**5**	**5**	**5**	**90**

DEVICE-SIDE MATRIX 2: Tethered of Wireless?
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
Wired	2	*	3	4	2	5	5	4	5	116
Wireless	**5**	*****	**5**	**5**	**5**	**4**	**3**	**5**	**2**	**141**

DEVICE-SIDE MATRIX 3: Wireless Medium
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
Infared	3	*	*	*	5	5	4	*	4	69
Ultrasound	2	*	*	*	3	5	4	*	5	64
Radio	**5**	*	*	*	**5**	**5**	**4**	*	**3**	**77**

DEVICE-SIDE MATRIX 4: Processing Location
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
Handheld Unit	4	*	4	*	*	5	3	3	*	86
Base Station	**5**	*****	**5**	*****	*****	**5**	**4**	**5**	*****	**106**
Device Being Interfaced (via Software)	5	*	5	*	*	2	5	4	*	102

DEVICE-SIDE MATRIX 5: Processor Architecture?
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	5	5	5	4	4	5	4	3	2	
ALTERNATIVES:										
A new USB chip	*	*	*	*	*	2	2	4	3	36
8051	*	*	*	*	*	5	5	3	5	64

DEVICE-SIDE MATRIX 6: First Port
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**3**	**2**	**4**	**5**	**5**	**3**	**2**	
ALTERNATIVES:										
MacADB	*	*	*	5	*	3	3	*	4	48
USB	*	*	*	5	*	2	2	*	3	36
RS-232	*****	*****	*****	**5**	*****	**5**	**5**	*****	**5**	**70**
AT	*	*	*	5	*	5	4	*	5	65

DEVICE-SIDE MATRIX 7: Second Port
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
MacADB	*	*	4	4	*	3	3	3	4	80
USB	*	*	4	5	*	2	2	5	3	79
RS-232	*	*	1	1	*	5	5	3	5	73
PC-AT	*****	*****	**5**	**5**	*****	**5**	**4**	**4**	**5**	**108**

DEVICE-SIDE MATRIX 8: AT Port Method
1=Low, 5=High, *=Not Applicable

CONSIDERATIONS:	PORTABILITY	HEALTH&SAFETY	ACCEPTABILITY	EFFECTIVENESS	MULTITASK|ABILITY	FEASIBILITY	SIMPLICITY	EXTENDABILITY	COST	TOTAL
WEIGHT [1-5]:	**5**	**5**	**5**	**4**	**4**	**5**	**4**	**3**	**2**	
ALTERNATIVES:										
Tap into an existing PC Keyboard Matrix	*	*	*	3	*	5	4	3	3	68
Use a RS232->AT keyboard "wedge"	*	*	*	4	*	5	5	3	1	72
Replicate PC-AT Protocol	*****	*****	*****	**5**	*****	**5**	**3**	**4**	**5**	**79**