The Design and Implementation of inTouch: A Distributed, Haptic Communication System

Victor C. Su

Bachelor of Science, Massachusetts Institute of Technology,
Cambridge, Massachusetts, June 1998

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

February 3, 1999

Author_____
Department of Electrical Engineering and Computer Science
February 3, 1999

Certified by_____
Hiroshi Ishii
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

The Design and Implementation of inTouch: A Distributed, Haptic Communication System

Victor C. Su

Submitted to the
Department of Electrical Engineering and Computer Science

February 3, 1999

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The inTouch project explores the notion of haptic interpersonal communication. Using bilateral force-feedback to couple two physically separated nodes, inTouch provides a channel for expression through touch. This interaction is derived from the concept of synchronized, distributed physical objects. This document explains the design and implementation of a self-contained, microcontroller-based version of inTouch. The structure of the system described here is intended to serve as an example of this concept as it relates to interpersonal communication and as a starting point to incorporating tactile channels into communication technologies.

Thesis Supervisor: Hiroshi Ishii
Title: Associate Professor of Media Arts and Sciences, MIT Media Laboratory

Acknowledgements

I would like to thank Scott Brave and Andrew Dahley for their original idea of inTouch as well as Phil Frei for his effort in designing the mechanical system. I am also grateful to Chris Leung and Rujira Hongladaromp for their work in the production run. Most of all, I would like to thank Professor Hiroshi Ishii for his support and encouragement.

Table of Contents

Table of Figures

1 Introduction

The goal of any communication technology is to bridge the distance between people by extending the senses through some transmission medium. Current communication technologies, such as the telephone or the videophone, appeal exclusively to the visual and auditory senses. Furthermore, work in the field of telepresence research has focused on the visual and auditory extension of space [10]. What has largely been unexplored, however, is the sense of touch, especially communication by touch. In everyday situations, touch plays an important role in the relations between people. A handshake, a pat on the back, and a warm embrace are all examples of physical expression of emotion. At times, these actions are more compelling and expressive than words. The goal of this project is thus to study new forms of interpersonal communication by touch, and how a simple implementation of this concept may serve as a starting point for incorporating tactile channels into future communication technologies.

This chapter will describe the conceptual framework behind inTouch along with some related work. Chapter 2 will introduce several versions of the inTouch system prior to the implementation of inTouch-3. Chapter 3 will examine the hardware development of inTouch-3 and will be followed by a discussion of the control algorithm in Chapter 4. Chapter 5 describes the production effort of 10 sets of this system. Chapter 6 introduces the design and implementation of the main accessory to the system, the modem interface unit. Chapter 7 describes several other accessories designed for the inTouch-3. Finally, Chapter 8 will outline several areas of future work and will be followed by the conclusion in Chapter 9.

1.1 Design Concepts

Haptic interpersonal communication aims to allow distant users to sense each other's presence. Ideally, all tactile information should be transmitted to achieve this, but this feat is currently beyond the state of the art. Instead, the concept of a "shared physical object," as illustrated in Figure 1.1, is used. If two people manipulate a common object,

then they can communicate hapticly and sense each other's presence, as shown in (a). If the object is instead separated and linked by some sort of communication technology, then the illusion of a unified object can be created, as shown in (b). This is the concept of a *synchronized, distributed physical object.*

Figure 1.1: Object-mediated communication

InTouch is an example of this concept and was designed by Media Lab graduate students Scott Brave and Andrew Dahley. A conceptual sketch is shown in Figure 1.2. The physical structure of inTouch consists of a set of three cylindrical, wooden rollers embedded in a metal base. The rollers are hapticly coupled so that each roller behaves as though it were physically linked to its corresponding roller on the other unit. Interactions between users can include passively feeling the presence of the other, moving the object cooperatively, or fighting over the state of the rollers.

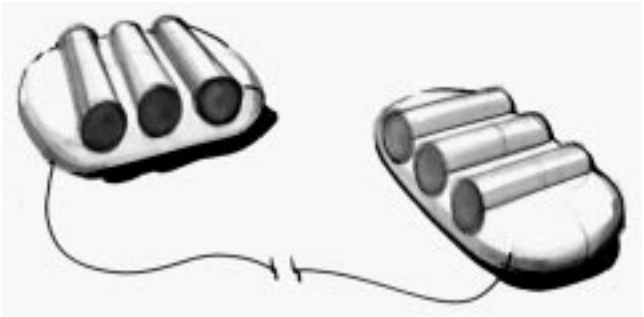


Figure 1.2: inTouch conceptual sketch

1.2 Related Work

Current haptic research deals with the creation of virtual objects with form, mass, and texture, and the force-feedback devices that allow them to be felt. An example is the *PHANTOM* by SensAble Technologies. In these systems, the force-feedback device acts

as a portal to the virtual world. InTouch, in contrast, does not link users to virtual worlds, but to each other.

Although inTouch is decidedly a device that links physical spaces to physical spaces, an example from the virtual world can still be referred to in this discussion. In most virtual environments, a single user interacts with virtual objects as though they had physical form. A scenario where more than one user interacts with the same virtual object, however, illustrates the concept of inTouch. In this case, the virtual environment serves as a mediator between physical spaces, connecting them like the inTouch.

In contrast to virtual reality research, tactile communication involves the extension of one's physical presence across distance. It is related to the field of tele-operation, which allows the operator to control a slave device such as a robot from a remote location. To facilitate the operator's control of the device, the forces exerted on it are reproduced at the point of control in a sort of feedback loop, giving the operator a tactile sense of the object under manipulation [7]. If the device under tele-operative control is not a slave robot but another operator, then a device similar in concept to the inTouch is created. Although tele-operation might be a plausible solution, it was not intended for the function of interpersonal communication, so a more elegant solution is desired.

Although few in number, there has been some previous work related to communication by touch. One such project is *Feather, Scent, and Shaker* [22] in which a pair of "shaker" objects is linked so that movement of one causes the other to vibrate and vice-versa.

Another project is *HandJive* [4], which allows the playing of haptic games through a pair of hand-held objects. The object consists of a shaft resembling a joystick, which when moved from side to side, causes a corresponding object to move up and down. The difference between *HandJive* and inTouch is that the *HandJive* devices are not tightly coupled. The input and output space exist on the same device, but are distinct. The design was so chosen to prevent "fighting" over the device. This "fighting," however, is fundamental to the concept behind inTouch.

A third project, which demonstrates the use of bilateral force-feedback in interpersonal communication, is *Kinesthetic Constructions* [21]. *Kinesthetic Constructions* employs a

network of large modern sculptures distributed around the world. Parts of each sculpture are connected to other sculptures hapticly.

2 Evolution of the Design

2.1 Mechanical Model: inTouch-0

Three versions of inTouch existed prior to the work documented in this thesis. The first version, inTouch-0, was a mechanical mock-up that served as a proof of concept and is shown in the illustration below. The rollers are connected together by stiff flex shafts.

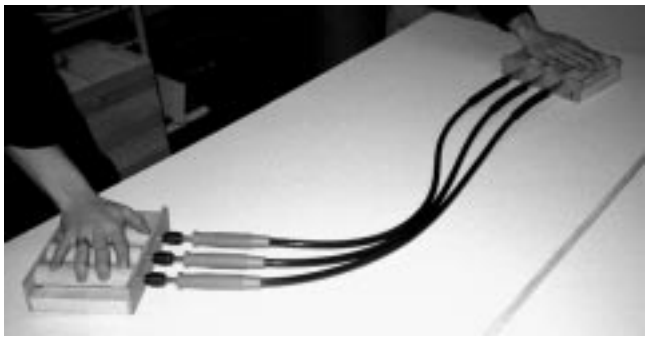


Figure 2.1: inTouch-0

2.2 Single Computer Version: inTouch-1

InTouch-1 developed as the first electronic prototype of the inTouch concept. The mechanical subsystem was designed by Phil Frei, a mechanical engineering student, and is shown below. The rollers are driven by precision, permanent-magnet DC motors made by Maxon, and are commonly used in force-feedback applications. Coupled to the drive shaft of each motor is a digital rotary encoder, which is used to track position. The encoder, made by Hewlett-Packard, has a resolution of 500 lines per revolution, which provides for position accuracy to about three-quarters of a degree of arc. Each roller is driven at a gear ratio of four to one, and thus 2000 lines of resolution are possible per revolution of the roller.

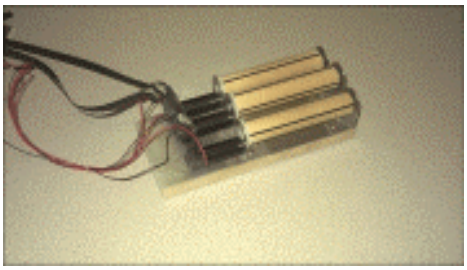


Figure 2.2: Mechanical subsystem of inTouch-1

The electrical subsystem consisted of a set of Impulse motor control boards, manufactured by Immersion Corporation. These motor control boards interfaced to the computer through PCI bus cards, and were programmed through C++ libraries provided by Immersion. The system architecture of inTouch-1 is shown here.

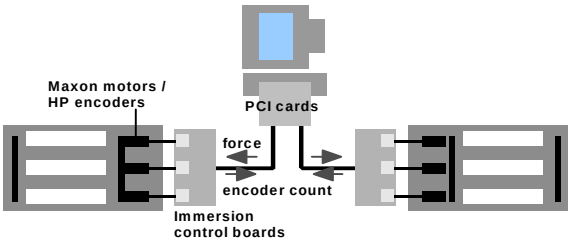


Figure 2.3: System architecture of inTouch-1

From this point forward, a *node* will refer to the combined electrical and mechanical subsystems, and represents an abstracted self-contained interaction unit. The diagram above thus illustrates that both nodes are controlled by algorithms running on a single computer, hence the name single computer version.

2.3 Distributed Computer Version: inTouch-2

For the third-generation version of inTouch, the mechanical subsystem was further refined by Phil Frei to the unit shown below. The motors have been encased in the left side of the unit and the shape made to conform to the placement of the rollers. This improves the aesthetic appearance of the unit and invites the touch.

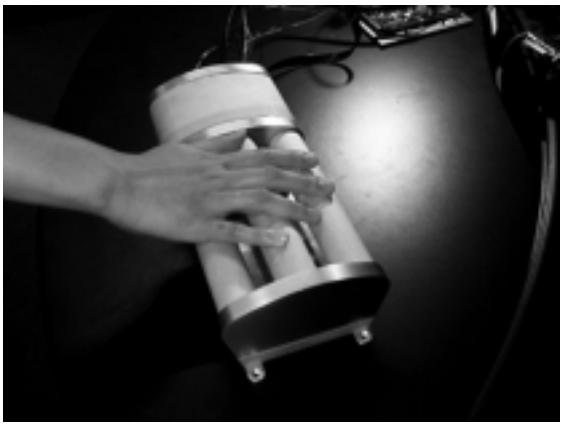


Figure 2.4: Mechanical subsystem of inTouch-2

The control algorithms of inTouch-1 were distributed symmetrically over two computers so that the nodes could communicate over a local area network by passing position information using universal datagram protocol (UDP). With this addition, communication latency became an issue, and because inTouch is a bilateral feedback system, obviously led to particularly undesirable effects such as system instability. These effects will be examined in the sections to follow. The distributed architecture system is shown here.

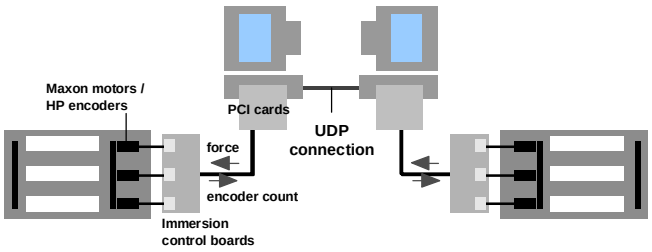


Figure 2.5: System architecture of inTouch-2

The system was tested using the Media Lab network and performed essentially identically to inTouch-1 as long as delay remained small. Typical network delays for transmission within the Media Lab are approximately two milliseconds. For moderate delays, compensation techniques such as velocity prediction permitted stable operation to approximately 12 milliseconds. For delays exceeding 12 milliseconds, noise present in the system prevented accurate prediction.

As an alternative means of compensation, low pass filtering was attempted. Since the interaction frequency (the frequency at which the rollers are moved forward and

backward) does not usually exceed five hertz, low pass filtering can reduce unwanted high frequency oscillation at the expense of tactile sensitivity. This filtering technique proved to be effective for delays up to 40-milliseconds [2].

2.4 Embedded Control Version: inTouch-3

This thesis documents the development of the fourth-generation inTouch system, inTouch-3. The goal of inTouch-3 is a self-contained, embedded control system, which implements the inTouch algorithm in custom hardware and firmware, and does not require the use of a computer. Communication between nodes complies with the RS-232C standard for serial communications, which permits future additions such as modules for modem communication as well as an interface to a computer-based simulator. The system architecture of inTouch-3 is shown below.

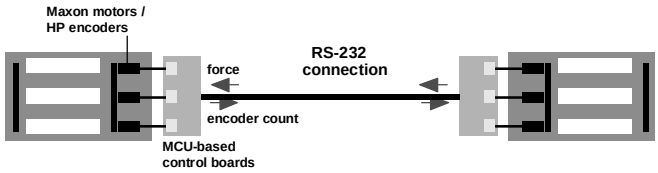


Figure 2.6: System architecture of inTouch-3

3 Hardware Implementation

3.1 Overview

The hardware implementation was accomplished first so that the control firmware could later be developed on a stable hardware platform. This thesis documents the development of the electrical subsystem of inTouch-3, since the mechanical subsystem is unchanged from inTouch-2. The functional blocks of the inTouch-3 system are shown below.

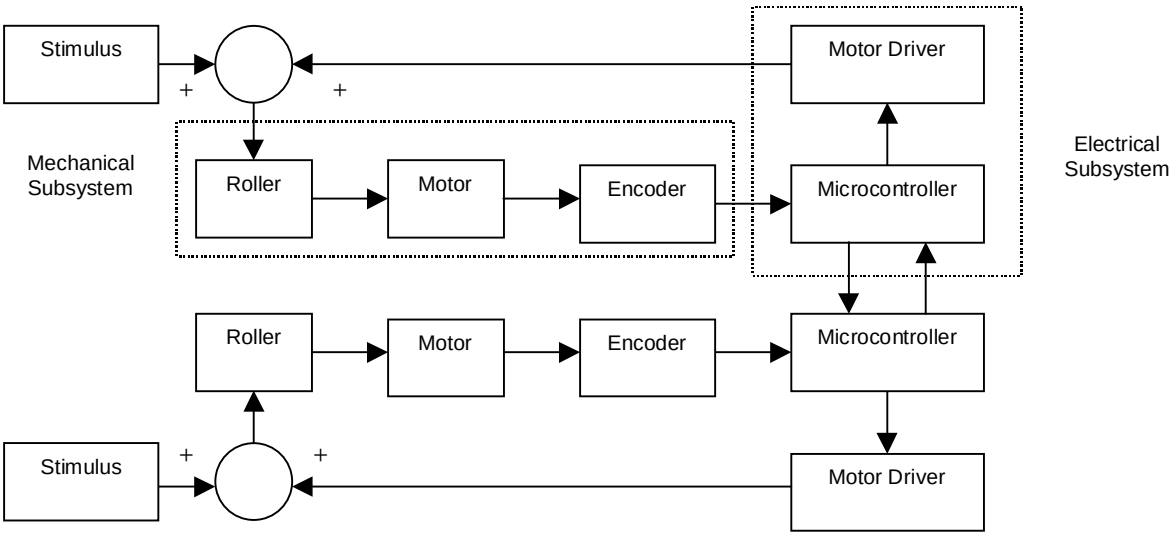


Figure 3.1: Functional diagram of the inTouch-3 system

The stimulus represents the users' input to the mechanical unit. This external torque acts on the rollers, which are coupled to the motors. The motor shaft is in turn coupled to a rotary encoder. The microcontroller, serving as the control core, reads the encoder position and transmits it to the remote node. The microcontroller then executes the inTouch synchronization algorithm on the encoder position received from the remote node as well as its own position and outputs an appropriate control signal, which the motor driver converts to current to drive the motor.

3.2 Microcontroller Selection

The PIC 16C73A microcontroller made by Microchip Technology, Inc., was selected for the controller core. This microcontroller can operate at high speed (20 MHz), and can thus provide ample processing power to implement the inTouch synchronization algorithm. Moreover, the PIC is inexpensive and has an efficient RISC instruction set and Harvard architecture. The microcontroller also offers several attractive features such as serial communications hardware and a full set of interrupts.

3.3 Encoder Reader

The first module to design in the inTouch-3 system was the rotary encoder reader used to track the position of the roller. The output of the encoder is a quadrature logic signal as shown in Figure 3.2. Note that the Phase A and Phase B waveforms differ in phase by 90 degrees. IN is an index pulse generated by the encoder once per revolution and can be used track absolute position. The index pulse is not used in inTouch since only relative positions are necessary, as will be apparent in the discussion of the algorithm.

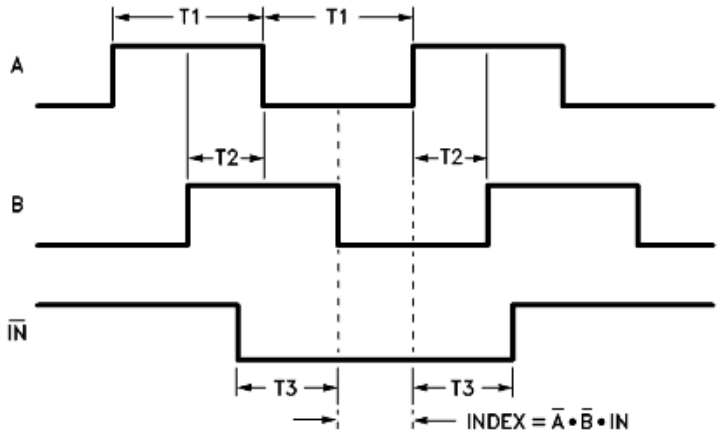


Figure 3.2: Quadrature waveform of encoder [16]

If the outputs of the encoder are considered as digital values, a sequence {00, 01, 11, 10, 00…} known as gray code is produced. The defining characteristic of gray code is that only one bit changes between code words, which prevents logic race conditions during code word transitions. The two sequences shown below illustrate the output from the encoder lines in each direction of rotation. The sequences should be read from top to bottom.

CW Rotation			CCW Rotation	
Bit A	Bit B		Bit A	Bit B
0	0		0	0
0	1		1	0
1	1		1	1
1	0		0	1
0	0		0	0

Table 3.1: Code sequences of quadrature encoder for each direction of rotation

Using the exclusive OR operator on bit A of one code word (for example the 0 in 01) with bit B of the following code word (the right 1 in 11), 1 is produced in one direction, while

0 is produced in the other. Thus, quadrature output is used to determine direction of shaft rotation.

The direction sensing described above may be implemented in software as firmware on the microcontroller or as hardware as a connection of two flip-flops. In the interest of reducing processing overhead on the microcontroller, the hardware implementation was chosen and is shown in Figure 3.3.

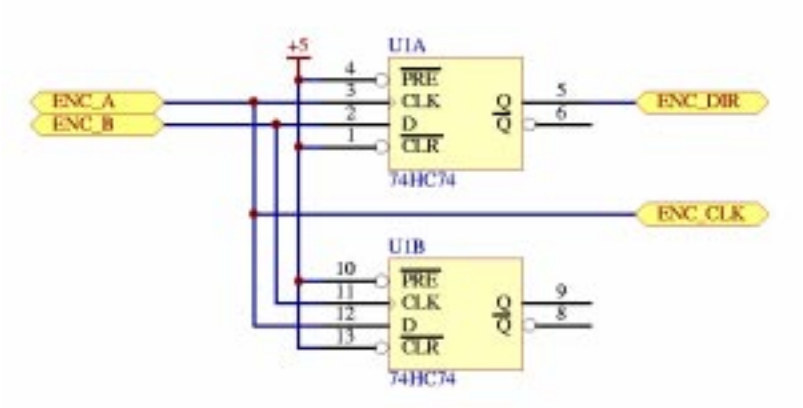


Figure 3.3: Schematic of direction sensing circuit

A digital counter maintains the encoder's position and is clocked from either phase A or phase B of the encoder. The choice is arbitrary. The output of the direction sensing circuit indicates the rotation direction and may be used to decide whether to increment or decrement the counter.

Again, the design choice here for the counter may be implementation in software or hardware. Necessary features of the counter include moderate to high resolution and bi-directional clocking. The latter condition is necessary since the inTouch application demands tracking of the roller in both directions. While up/down hardware counters are available (for example, the 74HC193), a software counter was chosen since it provides for easy changes to the resolution of the counter, and implementation of bidirectionality is simpler.

The software counter used in inTouch possesses only eight bits of resolution. To address the problem of counter roll-over when determining the change in position between counter values, a "shortest distance" method was used. The 8-bit counter takes on values from zero to 255 and is unsigned. Therefore, incrementing the counter at 255 results in a value of zero and decrementing it at zero results in a value of 255.

For discussion, assume a starting counter value of zero. If the next value is between zero and 127, the midpoint, then it is assumed that the encoder has moved in the positive direction from zero to that given value. If the next value is between 128 and 255, then it is assumed that the encoder has moved in the negative direction from 0 to that given value, rolling backwards to 255. To generalize for arbitrary starting values, the difference between counter values is used. If the difference between two successive counter values is less than 127, then the positive direction is assumed. If it is greater than 127, then the negative direction is assumed. Similarly, if the difference is greater than –127, then the negative direction is assumed. If it is less than –127, then the positive direction is assumed. This method is illustrated in Figure 3.4.

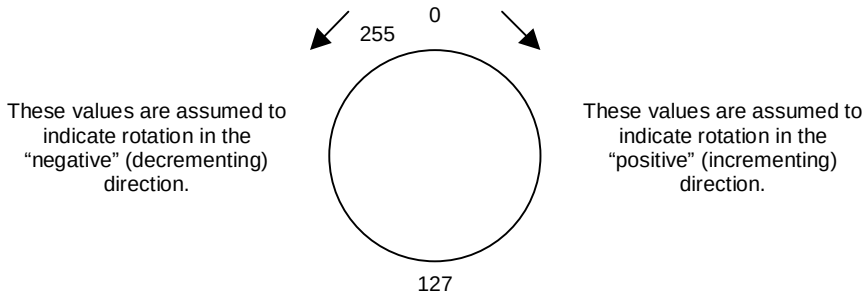


Figure 3.4: Accounting for counter roll-over

3.4 Encoder to Microcontroller Interface

There are two options for designing the interface between the encoder and the microcontroller. The microcontroller can poll the encoder to watch for changes in the encoder clock. This scheme, however, has the disadvantage that it requires substantial processing overhead, especially if the motor shaft is moving at high speed. In this case, the microcontroller must poll the encoder at least as fast as the expected rate of encoder change. Since the inTouch system uses high-resolution encoders at a moderate gear ratio, very small movements of the roller will result in encoder changes, so a high polling rate would be required.

An alternative solution to polling is to take advantage of the interrupt capabilities of the PIC microcontroller. The encoder clock may be connected to the external (EXT) interrupt of the microcontroller, which is executed on either the rising or falling transitions of the RB0 pin (the edge is software selectable). This allows the PIC to

execute other code in the time saved from not having to poll the encoder for changes. The interrupt service routine (ISR) for the EXT interrupt may thus examine the direction output from the encoder hardware and increment or decrement the software counter as appropriate.

An additional feature of the encoder to microcontroller interface that was not included in the final design is prescaling of the encoder clock. The objective of prescaling is to reduce the resolution of the encoder by eliminating clock pulses. Prescaling may be performed either in software by ignoring every other or every third clock pulse, for instance, or in hardware with a simple counter as shown in Figure 3.5. The outputs of the counter provide powers-of-two divisions of the encoder clock.

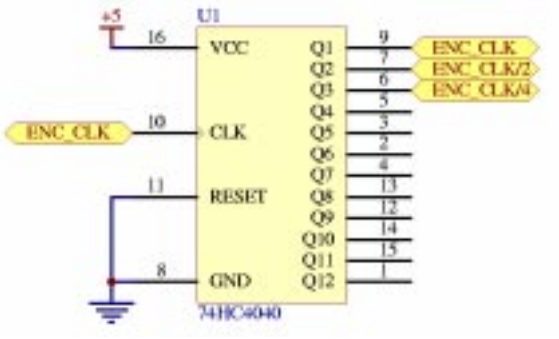


Figure 3.5: Hardware prescaling of the encoder clock

Prescaling might be used to reduce the resolution of the encoder when rotation speeds are too high. Assuming the interrupt clocking scheme described above, it is observed that there is a maximum clocking frequency that can be handled by the microcontroller. This is a function of both the instruction overhead necessary to service the interrupt as well as the execution time of the ISR code itself. Moreover, it is undesirable to have the interrupt service routine executing so frequently that other interrupts or the main loop of the program cannot be executed. Ultimately, the maximum clocking frequency is dependent upon the clock speed of the microcontroller. If prescaling is being used for this purpose, then hardware implementation is more sensible since reducing the encoder resolution prior to interfacing with the microcontroller will reduce execution frequency of the interrupt service routine.

3.5 Motor Controller

The next developmental block in the functional diagram is the motor driver hardware. Motor control may be divided into three main areas: torque control, velocity control, and position control. These three types of control are best implemented as closed loop systems, as the discussion here will show. Torque control requires current sensing hardware, which will be discussed in Section 3.5.2. Velocity control is typically performed by coupling a tachometer to the motor shaft, producing a voltage proportional to angular velocity. Position control may be achieved using a rotary encoder, which produces a quadrature pulse train as described in Section 3.3, or a simple potentiometer, which produces a varying voltage with respect to position.

InTouch requires both torque control for force-feedback and position control to maintain positional consistency of the rollers. Position control is achieved in the firmware of the microcontroller, using input from the encoder reader described in the last section, and will be discussed in more detail in sections to follow. Torque control will be examined here.

3.5.1 PWM Motor Drive

The initial design of the torque controller employed a scheme known as pulse-width modulation (PWM). In PWM, the pulse width, also known as the duty cycle, is varied between zero and 100 percent of the waveform's period. In this way, the time in which the waveform is logical HIGH is varied from zero, at zero duty cycle, to the whole period, at 100 percent duty cycle. Note that in the limiting cases zero and 100 percent, the waveform is constant. Figure 3.6 shows PWM waveforms for 25, 50 and 75 percent duty cycles. Another parameter of PWM is the switching frequency, which is the frequency at which a high pulse and a low pulse occur (with the exception of the limiting cases). The inverse of the frequency is the PWM period.

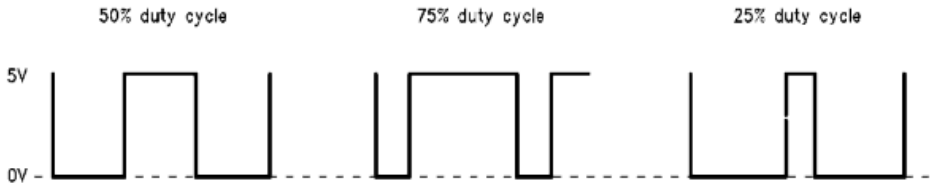


Figure 3.6: PWM waveforms for varying duty cycles [20].

If the PWM waveform is connected to the gate of a transistor, it may be used to vary the voltage at its terminals in proportion to the duty cycle, and do so very efficiently. Since the PIC microcontroller includes on-chip hardware PWM generators, they can be used for motor control with the added feature that essentially no software overhead is incurred in generating these waveforms.

3.5.2 H-Bridge Circuit

Although PWM can generate an analog voltage when used with a transistor as will be shown in the next section, the problem of how those transistors should be connected to drive motors must still be addressed. One desirable feature is that the circuit should only need one voltage supply, although motor drive should be possible in both directions. A configuration known as an H-bridge, Figure 3.7, provides the answer.

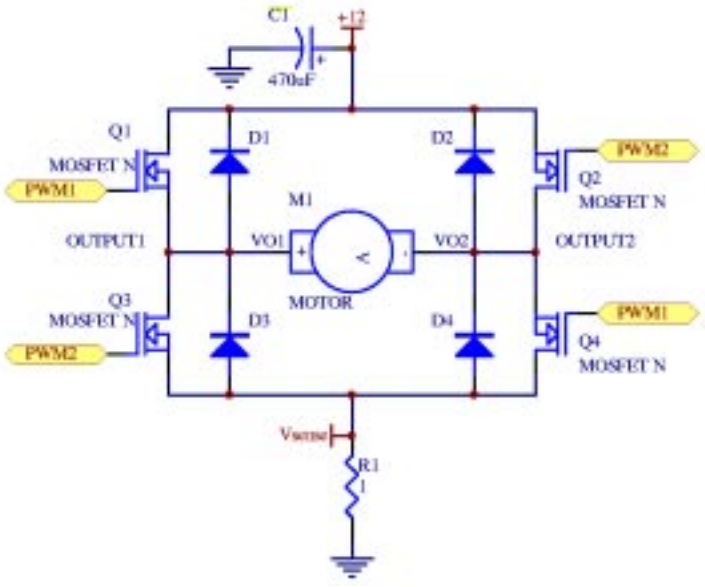


Figure 3.7: H-bridge circuit

Notice that when the upper left and the lower right transistors are switched on by applying sufficient voltage to PWM1, current flows through the motor in the left to right direction. When the upper right and the lower left transistors are switched on by applying sufficient voltage to PWM2, current flows through the motor in the right to left direction. Thus, by choosing the appropriate pair of transistors to drive, bidirectional motor drive can be achieved with only a single voltage supply. Because the H-bridge

circuit is used to drive inductive loads such as motors, protection diodes are added across the transistors to protect them from inductive reverse current. Note that the H-bridge shown above includes a current sensing resistor in the main current path. The resistor produces a voltage (V_{sense}) proportional to the current flowing through the bridge and thus can be used for current feedback control.

Rather than switching both transistors on each PWM cycle as described above, one terminal of the motor can be held at a fixed potential by turning on the upper transistor in that branch and turning on the lower transistor of the other branch during the on period and the upper transistor during the off period. The voltage at the other terminal of the motor is thus the time-average voltage as function of the duty cycle. The purpose of this method of drive is to prevent the motor from slipping during the PWM off time. During this portion of the waveform, both of the upper transistors in the H-bridge are switched on, creating a low impedance path between the motor terminals. This presents a particular problem for inTouch because the low impedance path means that the motor behaves as a generator under free rotation of the shaft. The induced voltage causes current to flow through this closed path around the motor terminals in a direction that resists the motion. To compensate for this "inductive friction" effect, the inTouch control algorithm, which will be discussed in Chapter 4, drives the motor slightly by an amount dependent upon velocity and thereby reduces the perceived friction.

Returning to Figure 3.7, it is seen that MOSFET's are typically used for H-bridge transistors. Field-effect transistors have essentially no gate current, and thus are easy to drive with logic. An enhancement mode *n*-channel MOSFET will turn on completely when its gate voltage is several volts above its source voltage. This does not present a problem for the transistors in the lower half of the bridge since their sources are referenced to ground. However, the sources of the upper pair of transistors in the H-bridge are referenced between ground and the supply voltage of the bridge, 12 volts, depending on whether those transistors are turned on. Therefore, under this configuration, the gates of the upper pair of transistors must be switched from a voltage above the supply voltage of the bridge to ensure proper operation.

One possible solution is to switch to an alternate configuration where *p*-channel MOSFET's rather than *n*-channel MOSFET's are used for the two transistors in the upper half of the bridge. A *p*-channel MOSFET turns on when there is no voltage on the gate (to be precise, when the gate voltage is close to the voltage of the drain), and turns

off when it is several volts above. (Note that for *p*-channel devices, the source voltage is more positive than the drain voltage and for *n*-channel devices, the source is more negative than the drain voltage.) With this modification, a logic inverter is needed to drive the *p*-channel MOSFET's from the PWM signal. This may be seen by noting that the *p*-channel MOSFET must be presented a logical LOW to turn on, while the *n*-channel MOSFET must be presented with a logical HIGH. One problem with this modification is that *p*-channel devices tend to exhibit poorer performance than *n*-channel devices, manifested as higher gate threshold voltage, higher on resistance, and lower saturation current. This behavior results of the particulars of semiconductor physics. Specifically, in *p*-channel devices, the carriers are quantum holes rather than electrons, which have lower mobility and shorter lifetime [8].

Another solution is to return to the original configuration of all *n*-channel devices, but switch the transistors with a voltage that is higher than the supply voltage of the H-bridge. This higher voltage can be generated using a device known as a charge pump, which will be discussed in Section 3.6. While the H-bridge can be built with discrete components as in Figure 3.6, sometimes it is more convenient to use integrated circuits that also provide additional features not easily achievable with discrete electronics. One example is the National Semiconductor LMD18200. The LMD18200 has a high current rating and built-in current sensing transistors to measure the amount of current flowing through the H-bridge. This device was chosen to implement the H-bridge in the initial tests described in the next section.

3.5.3 Initial Tests of Open-Loop Current Control

An initial test configuration was built and microcontroller code was written to sweep through the range of the duty cycle. The PIC was configured to produce 10-bit resolution PWM (1024 values) at a switching frequency of 19.53 kHz. The PWM output was connected to the LMD18200 to produce current at a supply voltage of 12 volts, the rating of the Maxon motor in the mechanical unit.

The results of the initial tests proved to be unsatisfactory as the PWM drive produced torque that was nonlinear with respect to duty cycle as shown in the figure here.

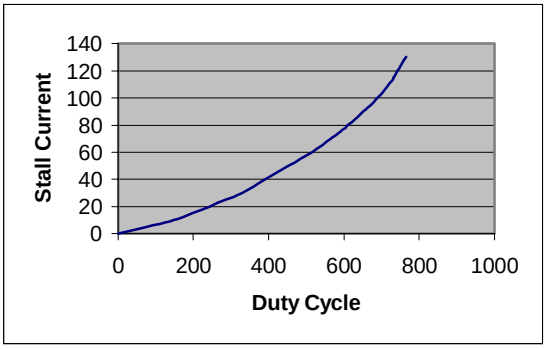


Figure 3.8: Torque as a function of duty in PWM drive

The stall current is the amount of current drawn by the motor with its rotors locked. The value shown in the graph is an 8-bit digital value acquired from the internal ADC of the microcontroller, mapped to an actual motor current range of 2.79 amperes. The largest data point is a value of 130, which corresponds to an actual current of about 1.4 amperes and is just slightly greater than the maximum continuous current limit of the Maxon motor. The current sense circuitry of the LMD18200 and the transfer function of the ADC are assumed to be linear and monotonic. The horizontal axis is the 10-bit duty cycle of the microcontroller's PWM generator.

The observed behavior is a result of several factors, but first the operation of the LMD18200 when used with PWM drive must be examined. The initial tests used sign/magnitude PWM drive as shown in Figure 3.9. In sign/magnitude drive, the magnitude is given by the PWM waveform, and the sign is given by a direction control line.

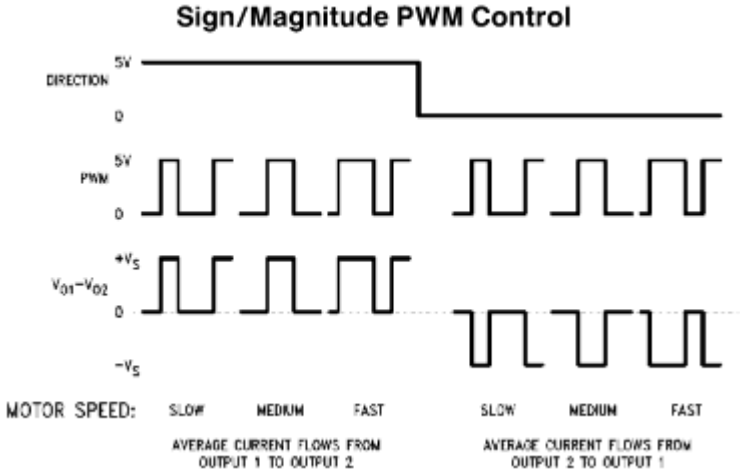


Figure 3.9: Sign/magnitude PWM drive [17]

Consider current flow from left to right through the motor in the H-bridge circuit of Figure 3.6. In this state, transistor Q1 is held on while Q2 and Q4 are alternately switched on and off. The net effect is that V_{O1} rises to 12 volts, while V_{O2} is the time average voltage of 12 volts and ground and is a function of the duty cycle. Since $V_{O1} \geq V_{O2}$, current will flow from output 1 to output 2. The situation is reversed for drive in the other direction.

The drive scheme described above suggests voltage drive rather than current drive and so the electrical dynamics of the motor must be considered. Modeling the motor as a first order system of an inductor in series with a resistor, constitutive relations yield:

$$v_a - v_{bemf} = i_a(R + sL) \tag{3.1}$$

where v_a is the applied voltage, v_{bemf} is the back EMF voltage, i_a is the applied current, R is the motor resistance, and L is the motor inductance. Calling the left hand term V_m and using the fact that motor torque is related to applied current by the motor's torque constant, K_m, Equation 3.1 becomes:

$$v_m = \tau K_m(R + sL) \tag{3.2}$$

Rearranging, the transfer function of voltage to torque is seen to be:

$$\frac{\tau}{v_m} = \frac{\frac{1}{KmL}}{s + \frac{R}{L}} \tag{3.3}$$

The solution of which is an exponential of time constant R/L. Moreover, the system tends to suppress high frequencies (as s approaches infinity, the transfer function goes to zero). Thus, waveforms with short duty cycles, which have greater high frequency content than waveforms with long duty cycles, are "filtered out" by the inductance of the motor and do not contribute to mechanical torque. At a certain "turn on" threshold, the inductance is overcome and the motor begins to move.

Another factor that may contribute to the nonlinearity, especially near zero, is the fact that the transistors in the H-bridge do not function as ideal switches. The voltage-current characteristic of a typical transistor shows that it goes out of conduction at a non-zero base or gate voltage. The initial tests used sign/magnitude modulation as described above. The shortcoming of this technique is that there is a dead band of current around zero caused by transistor cutoff. The contribution of these two factors may explain the observed results of the initial tests.

3.5.4 Closing the Loop: The PWM Servo Controller

Note that the actual current drawn by the motor is a function of the work performed. If the motor spins freely, the current will be small regardless of the duty cycle of the PWM driver. Increasing the duty cycle in this case will merely increase the rotational speed of the motor, up to the limit of the maximum speed of the motor. As the motor encounters more and more resistance, that is, as it performs an increasing amount of work, the current demanded similarly increases. The relationship of the duty cycle to the motor current is only consistent when the load is constant. If the load varies, then the motor current tends to follow the changes in the load. Therefore, open loop control of the current is an unsatisfactory solution for a force-feedback application such as the inTouch system, where precise torque and thus current control is necessary.

A possible solution to this problem is to use the current sense feedback output of the H-bridge to close the loop and adjust the duty cycle dynamically to achieve the desired current. In this way, current drive is achieved, since feedback eliminates the effect of the motor dynamics of Equation 3.1. The initial attempt to implement current control used feedback from the analog-to-digital converter of the PIC microcontroller and a proportional-accumulator-difference (PAD) filter to produce the output. The PAD is essentially a discrete-time PID filter, which will be examined in greater detail in Chapter 4. For now it suffices to know that the PAD algorithm works by examining the difference between a desired output and the actual output in determining how to adjust the output to achieve zero difference, or error. This type of controller is known as a PWM servo controller.

One reason for the failure of the PWM servo controller was that the internal ADC of the microcontroller, used for current sensing, possessed only 8-bit resolution, which most likely resulted in large amount of quantization error in this application. This problem could have been remedied by an external higher-resolution analog-to-digital converter such as the Linear Technologies LTC1298 at the expense of added complexity and hardware cost.

Another reason for the failure was calculation error in the PAD algorithm. The discrete-time algorithm approximates its continuous time counterpart, but a high sampling rate must be used to minimize the error. This was not possible since the microcontroller needed to execute the inTouch synchronization algorithm simultaneously. Two possible solutions may have been to use a dedicated processor or

shift current feedback control to the continuous-time domain by implementing it in analog circuitry. The latter solution will be examined in Chapter 8.

A third reason for the failure of the PWM servo controller code was that sign/magnitude PWM modulation used for motor drive did not have good control around the zero current point. The reasons for this problem were discussed in the last section. The importance of near-zero current control will be more apparent in the discussion of interactions in Chapter 4. For now, assume that the inTouch interaction requires this type of control for optimal performance. One solution to this problem is to use a technique known as 50 percent modulation, or locked anti-phase modulation, as illustrated in Figure 3.10.

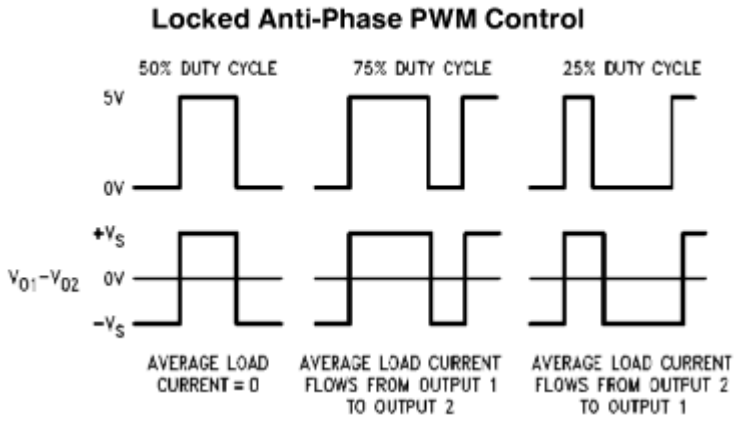


Figure 3.10: Locked anti-phase PWM modulation [17]

In locked anti-phase modulation, the diametrically opposite pairs of transistors are driven in alternation. If the duty cycle is 50 percent, the time-average voltage at each terminal of the motor is the same, and so the time-average current over one period is zero and the motor remains stationary. If the duty cycle is shifted either above or below 50 percent, the average current will be non-zero and will flow in the appropriate direction, as illustrated in the figure above. The advantage of locked anti-phase modulation is that only one control pin is necessary as both sign and magnitude information is contained in the waveform. Moreover, this form of modulation has improved near-zero current control over sign/magnitude modulation and exhibits no current dead band since the transistors are on and switching even though the motor is stationary. The scheme is recommend for "applications where fast dynamic control of inertial loads (i.e., the rapid reversal of the direction of rotation of a motor) is important

... [since] the 'regeneration' of net average power from the load back to the supply [is] able to take place [20]." Thus, it is particularly appropriate for inTouch.

3.5.5 The Commercial Solution

In the interests of saving time, commercially available motor drivers were ultimately used. The 4212Z servo amplifier, made by Copley Controls Corporation, was selected and configured for torque feedback control. This particular amplifier uses locked anti-phase PWM modulation for good near-zero current control and no current dead band. Current is also automatically adjusted through feedback and controlled by an analog voltage. The addition of these amplifiers necessitated slight modifications to the controller board circuitry to accommodate the analog control signals. The modified block diagram in shown in Figure 3.11.

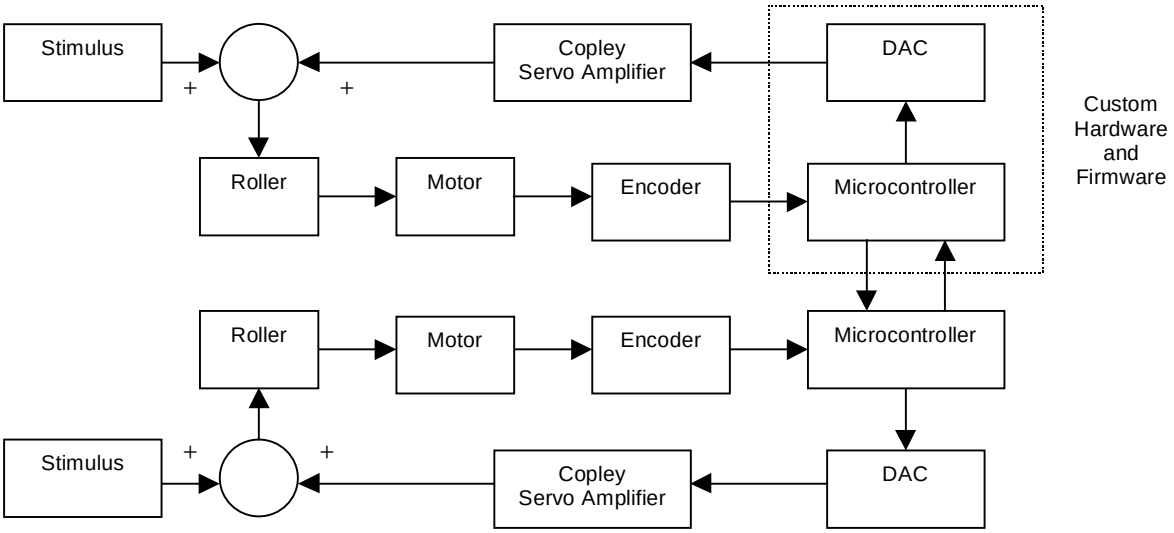


Figure 3.11: Modified functional block diagram of inTouch-3 system

Note in the figure that the motor driver block has been replaced by the digital-to-analog converter (DAC) block and the servo amplifier block. The implementation of the DAC block will be described in the next section. The servo amplifiers designated in the figure are the Copley motor drivers. No changes were made to the Copley amplifiers except that the transconductance was changed to 0.7 amperes/volt by replacing one of the configuration resistors. This setting indicates that 0.7 amperes of current are produced for every volt of control signal. The reader is referred to the 4212Z user's manual, available from Copley Controls Corporation, for the details of operation.

The Copley servo amplifiers can be configured to work with motors of varying inductance. Since the Maxon motors used for the inTouch were on the smaller side, a power inductor was added in series with the motor to increase the total inductance. The addition of the inductor satisfied the servo amplifier's minimum inductance rating and also reduced inductive heating in the motors caused by the locked anti-phase modulation, which might shorten motor operating life.

The unit that is comprised of the DAC, the microcontroller, and the encoder reader, will from this point forward be termed the *controller board*. This board, along with the Copley servo amplifier will be termed the *controller unit*.

3.6 Digital-to-Analog Converter

The Copley amplifiers accept industry-standard analog control signals ranging from –10 volts to 10 volts. With the transconductance set at 0.7 amperes/volt as described in the last section, about 1.8 volts are necessary to produce the 1.25 amperes at which the Maxon motors are rated. An external DAC, the Maxim MAX504, was selected to produce the control voltage. The MAX504 has 10 bits of resolution and is interfaced through serial peripheral interface (SPI).

SPI describes a synchronous serial port. Data is sent to the device from the microcontoller one bit at a time. Each bit is distinguished by a clock pulse from the microcontroller, hence the term synchronous. The DAC accepts a 10-bit digital value that is transferred by SPI and which sets the analog output voltage. The DAC can be configured to produce a bipolar signal, although this reduces the resolution from 10 bits in one polarity to 9 bits in two. In order to produce bipolar voltages, however, the DAC must be powered by bipolar voltage supplies. The problem thus became how to convert the five volt logic supply into negative five volts to power the DAC.

The solution was to use a charge pump converter, the Linear Technologies LTC1044 as shown in Figure 3.12. The charge pump converter works by accumulating charge in a bucket capacitor, C12, then transfering it to a reservoir capacitor, C13. During the first half of the switching cycle, the bucket capacitor is connected to V+, charging it to that voltage. During the second half of the switching cycle, the positive terminal of the

bucket capacitor is connected to ground, while the negative terminal is connected to V_{OUT}, thus inverting the voltage V+.

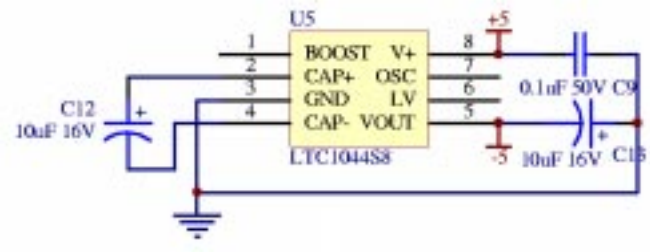


Figure 3.12: Charge pump negative voltage converter

With 5 volt and –5 volt supplies, the MAX504 can produce analog voltages in the range of –2.048 to +2.048 volts, over which the 10 bits of resolution are mapped. Thus, by writing the appropriate value into the DAC, the microcontroller can set the drive current to the motor. Feedback is performed in hardware by the servo amplifier to maintain the proper current regardless of the load on the motor.

3.7 Data Communications Interface

The microcontroller communicates with its remote counterpart using the RS-232C standard for serial communication. The RS-232C standard specifies electrical parameters as well as the data format. The most common data format consists of a start bit, a logical HIGH to LOW transition, followed by 8 data bits, followed by a stop bit, a LOW to HIGH transition. This format is known as 8N1, which represents eight data bits, no parity, and one stop bit. The number of data bits can vary from 5 to 9 and more than one stop bit may be used. Another important parameter of RS-232 is the baud rate, which is the number of clock periods per second. InTouch uses 8N1 data format at a baud rate of 19200. Thus, the bit period is approximately 52.08 µs and one byte requires 520.8µs to be transmitted.

The RS-232C standard uses what is known as non-return-to-zero (NRZ) bipolar voltage signaling. A logical LOW is indicated by a voltage of +5 to +15 volts, while a logical HIGH is indicated by a voltage of –5 to –15 volts. The driver must be able to operate into loads of 3000 to 7000 ohms with a slew rate less than 30V/ µs and with the ability to withstand short circuits. The receiver must present a load of 3000 to 7000 ohms and

convert an input of +3 to +25V to logical LOW and –3 to –25 volts to logical HIGH [8]. An example waveform is shown in the figure below.

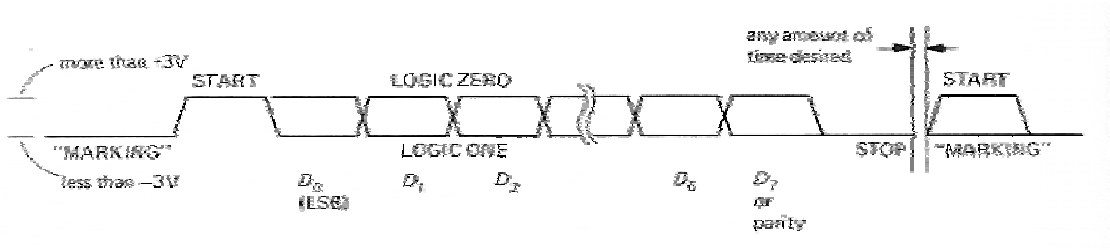


Figure 3.13: RS-232 serial byte waveform [8]

The data format is handled by the microcontroller in firmware or internal hardware. The PIC 16C73A contains a module known as a universal synchronous/asynchronous receiver transmitter (USART), which handles generating the baud clock, serializing data for transmission, and shifting incoming data into a receive data register. For microcontrollers that do not possess internal USART's, an external USART or UART (same as USART but only asynchronous mode) may be used. Alternatively, RS-232 data format may be implemented in firmware. The PIC C compiler from Custom Computer Services, Inc., for instance, includes libraries to facilitate serial communications.

The one disadvantage of software implementations of RS-232 is the processing overhead needed to poll for and receive data, although a method of using interrupts to eliminate the need for polling will be described here. The internal USART of the PIC 16C73A includes associated receive data and transmit data interrupts. The receive data interrupt eliminates the need for data polling. The following interrupt service routine illustrates how to use the interrupt to receive data as it arrives:

```
#INT_RDA
void Service_RDA() {
   Process_Serial(getc());
}
```

`Process_Serial()` is assumed to be a procedure which processes serial data after it is received. `Getc()` is a compiler command that copies data from the receive data register into the working register, so that it may be passed as an argument to `Process_Serial()`. In order to activate the receive data interrupt, it must be enabled either by setting the appropriate bit in the PIR1 register of the PIC or by using the compiler's `enable_interrupts()` function. In addition, the GLOBAL interrupt must be

enabled to activate the interrupt handler. The reader is referred to the PIC 16C73A user's manual available from Microchip Technology and the C compiler manual available from Custom Computer Services for more information.

For microcontrollers that do not possess serial hardware, but do possess the external (EXT) interrupt described in Section 3.4, the following technique may be used. Recall from the discussion above that the RS-232 start bit is a transition from logical high to logical low. Thus to detect the start bit, the EXT interrupt should be configured to trigger on the falling edge and the following interrupt service routine may be used:

```
#INT_EXT
void Service_EXT() {
   Process_Serial(getc());
}
```

In the code segment above, `Process_Serial()` is again a procedure to process serial data after it has been received. However, `getc()` in this case is a compiler-generated software routine that receives RS-232 data. As an implementation note, the INTF bit of the INTCON register should be cleared prior to activating the interrupt handler or there is a possibility that the EXT interrupt service routine may be executed immediately even though no data is being received. Since `getc()` blocks to receive data, execution will stop at this point waiting for data that is not present.

The transmit data interrupt is not used in the inTouch application, but indicates when the USART is ready to accept more data to transmit. It is useful in applications where streaming data must be transmitted at the maximum capacity of the channel, since data can be transmitted consecutively with minimal delay between bytes.

To implement RS-232 electrical specifications, a level shifter was used. The Maxim MAX233 provides a convenient minimum-component solution. Charge pumps are traditionally used to produce the bipolar voltage levels of RS-232. The MAX233 conveniently incorporates the entire charge pump internally so that no external capacitors are needed. Although the MAX233 ensures compliance with the RS-232 specifications, it may not be necessary in all applications, especially those in which cost is an issue and where cable lengths are short. The RS-232 standard works for cable lengths of at least 50 feet [8]. The actual limitation is a function of the noise present and the impedance characteristics of the cable and may be maximized by using twisted-pair cable. For inTouch, the objective was to maximize the cable length so that

the nodes could be separated by large distances. Thus, a fully compliant transceiver such as the MAX233 was needed.

For extended distances, a differential signaling protocol such as RS-485 may be used. In contrast to RS-232 where all signals are referenced to ground, differential signaling protocols use the voltage difference between two lines. This scheme has better noise immunity since noise present in the system will most likely effect both lines (i.e. common mode) and will cancel when the difference is taken (i.e. differential mode). Strictly speaking, RS-485 only defines the electrical parameters. The data format is arbitrary, so RS-232 8N1 data format may be used, for example. The other feature of RS-485 is that it supports multi-drop configurations, so that many nodes may be connected to the same cable, although in this case more complex network arbitration may need to be used. The reader is referred to National Semiconductor application note 1057 for more information about RS-485.

3.8 Power Supply

Most of the power consumed by the inTouch system is in the form of current used to drive the Maxon motors. Under normal interaction patterns, the motors operate only slightly above their no-load current. Maximum current will occur if the users twist the rollers apart in opposite directions until the current limit imposed by the inTouch algorithm takes effect. If all three rollers are subjected to this condition, the approximate power consumption will be (3)(1.25A)(12V) = 45 watts, assuming 100 percent efficiency. In reality, efficiency is less than 100 percent, especially since the efficiency of the Maxon motor is only approximately 80 percent. Moreover, the Copley servo amplifiers require 24 volt supplies to produce 12 volt bidirectional drive. Inevitably, some loss occurs in the voltage conversion.

Although this may be the maximum *continuous* current, transient currents produced, for example, when the motor abruptly reverses direction may be much higher. Under these conditions, the power supply must be able to source and sink power rapidly. The power supply must be able to absorb the energy stored in the inductance of the load quickly when a direction reversal, and thus a current reversal, is demanded.

Initial tests used 60 watt power supplies that shut down when large transient currents such as those described above were produced. To solve these problems, a 150 watt

power supply was selected. The rating is arbitrary; 150 watts was selected for reasons of availability and cost.

The electronics, in contrast, operate on five volt supply with low current. A simple method of producing low DC voltages from higher DC voltages is with a linear regulator such as the LM340. The problem with linear regulators is that their efficiency is usually no better than 50 to 60 percent. The resistive loss manifests itself as radiated heat and thus linear regulators must be heat sunk when operating at high power. Recall that power is the product of current and voltage. Thus, high heat dissipation occurs when current is high or voltage is high or both. Recall that the input to the regulator would be 24 volts, assuming that the same power supply for the Copley amplifiers was used for the electronics. This condition would mean that the regulator would need to accommodate a 19 volt voltage drop, so heat dissipation would be enormous even at low currents.

The solution was to implement a switching regulator, which can have an efficiency approaching 90 percent. The National Semiconductor LM2574 is a simple switching regulator that requires a minimum number of external components. The operating principle of the switching regulator is that energy is stored in the magnetic flux of the inductor and switched into the load. Since flux and not charge, as in the charge pump converter, is used to store energy, the voltage may be higher, lower, or the inverse of the input voltage.

A step-down (buck mode) regulator, as shown below, converts a high voltage to a low voltage. The output voltage is compared to an internal voltage reference. If the output falls below the reference, an oscillator turns on, which drives current into the inductor until the output voltage rises above the reference. The output capacitor, C5, forms an RLC filter with the inductor and the load to remove the switching frequency, leaving DC voltage. Since the voltage of the inductor is not constrained, the catch diode, D2, turns on when the OUTPUT pin drops to -0.4 volts to protect the regulator. Diode D1 protects the regulator from negative input voltages caused by reversed connections.

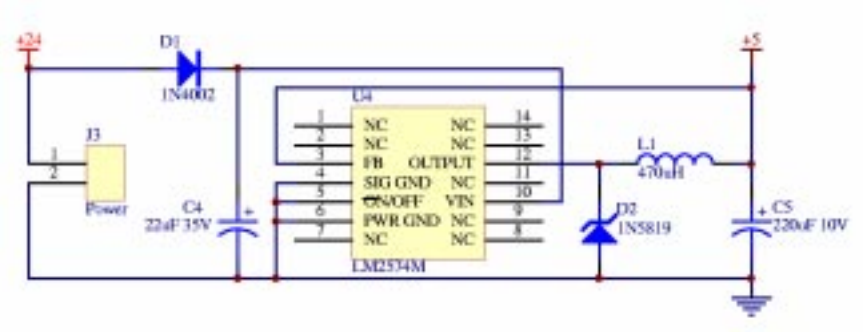


Figure 3.14: Step-down switching regulator circuit

3.9 Mechanical Unit Interface

The mechanical units were initially designed to house the electronics in a small space in the base. After the electronics were developed, however, this space was found to be too small. The solution was to place the electronics in a separate control box away from the mechanical unit, presumably on the floor. Only encoder and motor drive connections would need to be made between the control box and the mechanical unit, however, they would need to be separated by a moderate distance (approximately six to ten feet). This did not present a problem with the motor drive connections; larger gauge wire would suffice to carry the needed maximum current over greater distances. However, the encoder outputs would not operate reliably over such lengths of cable under the present configuration.

The problem was that the encoder had open-collector outputs that needed pull-up resistors to produce valid logic levels. Open-collector interfaces do not work well over long cable runs that are subject to environmental noise and signal reflections. The solution was to use a line driver, the 74AC244, near the encoders to drive current through the cable and a schmitt trigger, the 74HC14, at the other end to recover clean logic signals from the noisy received signals through hysteresis. The end of the encoder extension cable is AC terminated using a resistor and a capacitor chosen to match the characteristic impedance of twisted pair cable, 100 ohms.

To make this possible however, an auxiliary circuit containing the line driver would need to be placed in the base. Power could be supplied by the control box through the same cable that carried the encoder signals from the mechanical unit. Figure 3.15 shows the configuration for one encoder. Note that the line driver is selected from the

AC logic family, which has higher source current capability. This is particularly important for overcoming cable capacitance when driving long cables.

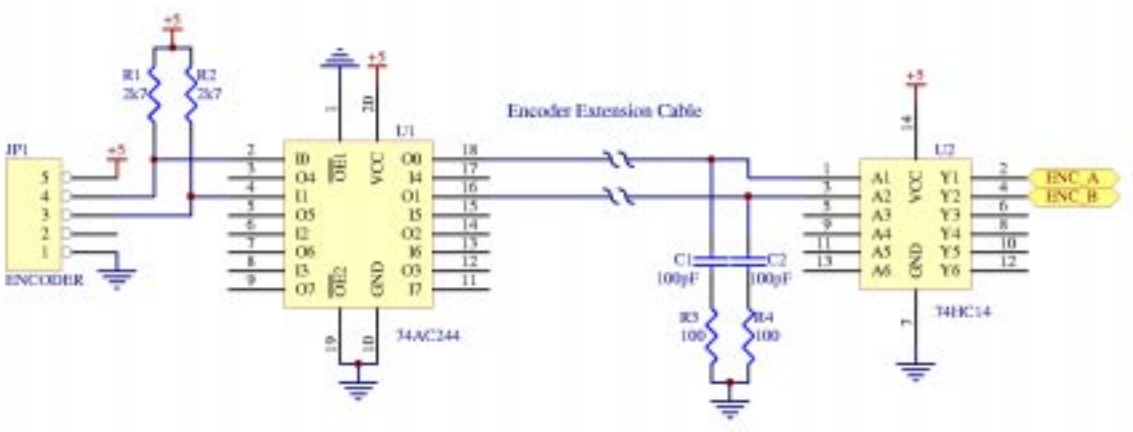


Figure 3.15: Extending the encoder cable length

3.10 Fault Recovery and Troubleshooting

Several features were included both in hardware and firmware to make inTouch-3 more robust and easier to troubleshoot. In hardware, several status LED's were included to aid problem diagnosis. For example, LED's were added to verify operation of the five volt logic supply, operation of the encoder, serial data reception and transmission, and a general status LED to indicate proper execution of firmware.

Several features of the PIC microcontroller facilitate fault isolation and recovery. The microcontroller contains an internal watchdog timer that runs independently of the system oscillator. The purpose of the watchdog timer is to automatically reset the PIC should execution stop for any reason. Also included in the firmware are several initialization checks to verify proper operation of the DAC and the servo amplifier. If a fault occurs during initialization, an error code is transmitted on the serial output to aid in repair. Verification is also performed during regular operation should any fault arise in that case. Initialization and operation faults are handled by stalling the microcontroller until the watchdog timer automatically restarts the device.

4 The inTouch Control Algorithm

Now that the hardware platform has been developed, the inTouch control algorithm can be implemented in microcontroller firmware. The development of the algorithm will begin by examination of the fundamental principles.

4.1 Mechanical Subsystem

The mechanical subsystem of inTouch consists of a set of three wooden rollers driven by DC motors. Recall that the motor and roller exhibit a certain amount of load damping, B_L, and rotational inertia, J_T. The rollers experience torque, T_m, produced by the motor, and disturbance torque, T_d, produced by the user, both of which influence the shaft position, θ. Newton's law gives:

$$J_T\ddot{\theta} = T_m + T_d - B_L\dot{\theta} \tag{4.1}$$

Taking Laplace transforms yields:

$$J_T s^2\theta = T_m + T_d - B_L s\theta \tag{4.2}$$

This result forms the core of the block diagram shown below. An integrator block is attached to convert angular velocity to angular position, θ. The motor torque, T_m, is related to the drive current by the torque constant, K_m, since current drive is used.

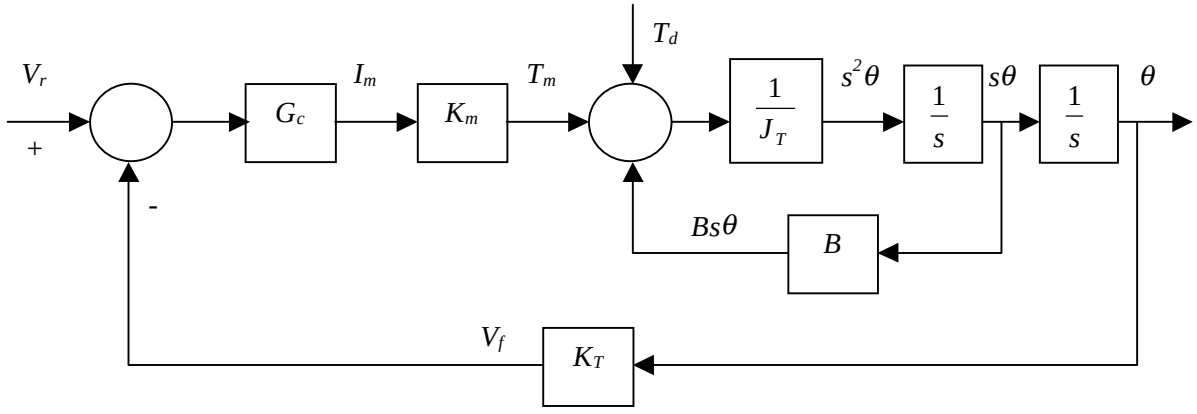


Figure 4.1: Block diagram of mechanical subsystem and controller

4.2 Control Loop

Since the mechanical subsystem is subject to physical limitations in bandwidth and speed, a control loop is formed around it to compensate for these characteristics. The addition of a feedback block, K_T, a position transducer in this case, and a control block, G_C, forms the control loop. The reference voltage, V_r, sets the desired position and is compared to the feedback voltage, V_f, in the controller to command the mechanical subsystem. The fact that the system responds to error, the difference between these two voltages, is what characterizes it as a feedback system.

Position consistency is achieved by coupling the rollers with a critically damped, computational spring. The inTouch control algorithm is the computational equivalent of the flex shafts of inTouch-0, upon which the system is modeled. Recall from mechanics that a spring exerts an opposing force of magnitude proportional to the displacement. A damped spring can be modeled as an ideal spring in addition to a shock absorber, which exerts a damping force that is proportional to velocity. Equation 4.3 describes the damped spring system.

$$F = -K\Delta x - B\Delta \dot{x} \tag{4.3}$$

where F is the force exerted, K is the spring constant, B is the damping coefficient of the shock absorber, and Δx is the displacement. Since the flex shafts of inTouch-0 are rotary springs which oppose angular movement, the equations become:

$$\tau_0 = K(\theta_1 - \theta_0) + B(\dot{\theta}_1 - \dot{\theta}_0) \tag{4.4}$$

$$\tau_1 = K(\theta_0 - \theta_1) + B(\dot{\theta}_0 - \dot{\theta}_1) \tag{4.5}$$

where

$\theta_{0/1}$	angular positions of the two coupled rollers
$\tau_{0/1}$	torque to exert on the corresponding roller
K	spring constant
B	damping coefficient

Note that the angular positions are defined in such a way that the resulting torque opposes the movement of the roller (the negative sign of Equation 4.3 has been distributed).

In reality, a damped spring model exhibits mechanical friction, which is velocity-dependent. To compensate for this fact, an anti-damping force may be added to the system so that the perceived friction is lowered. Adding this term, the equations above become:

$$\tau_0 = K(\theta_1 - \theta_0) + B(\dot{\theta}_1 - \dot{\theta}_0) + B_{fricit}\dot{\theta}_0 \quad (4.6)$$

$$\tau_1 = K(\theta_0 - \theta_1) + B(\dot{\theta}_0 - \dot{\theta}_1) + B_{fricit}\dot{\theta}_1 \quad (4.7)$$

where B_{fricit} is the friction compensation coefficient. Equations 4.6 and 4.7 represent the action of the controller.

The algorithm attempts to maintain this mechanical to computational equivalence insofar as the electrical system will permit, with a few notable exceptions. Two instances in which this equivalence breaks down are the limitations of finite gain and finite motor torque.

The mechanical model exhibits an extremely high spring constant determined by the torsional rigidity of the flex shafts and a large maximum force deliverable before breaking. The response of the shafts is fast with infinitesimal overshoot and rapid settling time, corresponding to a critically-damped system. While the computational model may approach the performance of the mechanical model by increases in the system gain, corresponding to the high spring constant, doing so may cause adverse effects such as instability. While such instabilities may be prevented through appropriate compensation techniques, the maximum gain will be finite and less than the "gain" of the mechanical model.

Furthermore, the maximum output torque of the motor, which is finite, determines the maximum force deliverable by the computational system. Therefore, a limit must be imposed on the maximum current applied to the motor, and thus the maximum torque produced by it. If the rollers are pulled apart, the computational system will follow the spring model until the torque limit of the motor is reached. At that point, the torque is held constant for all displacements beyond the limit. One disadvantage of the torque limit is that it breaks the illusion of a synchronized, distributed physical object.

4.3 PID Controller

The control block of the feedback system contains a proportional-plus-integral-plus-derivative (PID) filter, which receives the error as an input and produces an appropriate output. The theory behind the PID controller will be examined first.

4.3.1 Theory

The primary purpose of the PID controller is to compensate the control loop for the dynamics of the mechanical system. As seen in Figure 4.1, the mechanical system is second-order between torque and position and includes two mechanical parameters, the damping and the rotational inertia. When properly tuned, the PID controller produces a system response that is critically damped and compensates for the limitations of the mechanical system. To analyze system response, performance metrics for transient response and stability must be introduced.

System performance is usually characterized by three parameters: maximum overshoot, rise time, and settling time. Maximum overshoot, M_p, is defined as the peak value of the unit step response measured from unity. The amount of overshoot relates to the relative stability of the system [6]. The rise time, t_r, is the time required for the response to rise from ten to ninety percent of the final value. The settling time, t_s, is the time required for the response to reach and stay within two percent of the final value [6]. A critically damped system exhibits the minimum possible rise time that maintains zero overshoot and damped oscillations. Figure 4.2 illustrates the transient response attributes of a system and Figure 4.3 shows the unit step response of a critically damped system.

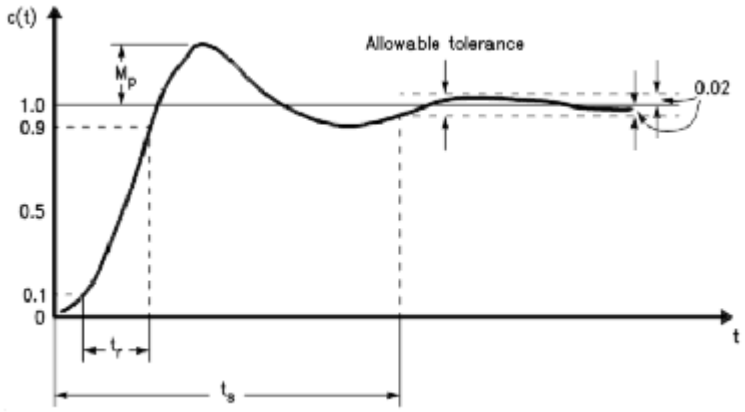


Figure 4.2: System transient response attributes [6]

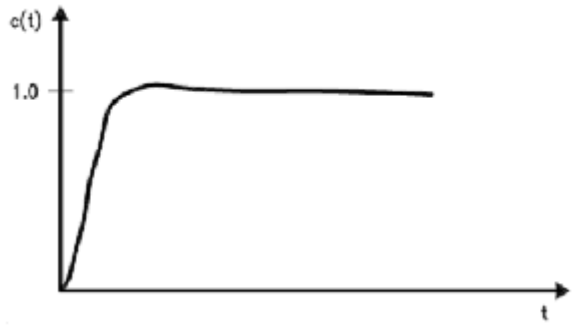


Figure 4.3: Unit step response of a critically damped system [6]

To assess system stability, a number of techniques may be utilized. Two common metrics, gain margin and phase margin, are used to describe the properties of the open-loop frequency response. The gain margin (g.m.) is "the amount of increase in gain where the phase is –180° that produces instability [6]." The phase margin (p.m.) is "the amount of negative phase shift that when added at the frequency where the magnitude is unity will produce instability [6]." Two other notable points are the magnitude crossover frequency and the phase crossover frequency. The magnitude crossover frequency, ω_c, occurs where the open-loop gain magnitude is unity. Its reciprocal is roughly proportional to the rise time for systems with frequency-independent feedback paths [6]. The phase crossover frequency, ω_ϕ, occurs where the phase of the open-loop frequency response is –180°. These measures of system stability are shown in the Bode plots of the open-loop transfer function $L(s)$ in Figure 4.4.

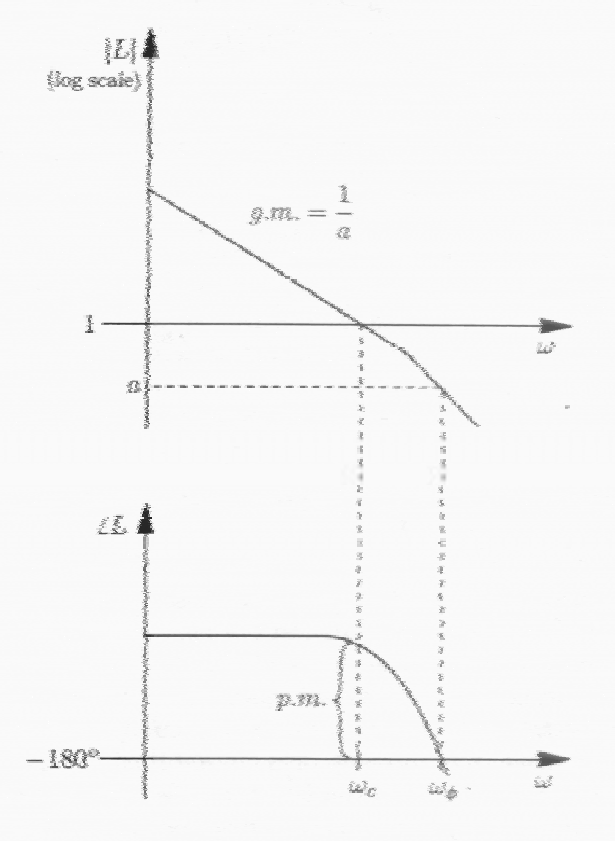


Figure 4.4: Gain and phase margins of open-loop transfer function [6]

To achieve optimal performance in terms of these measures of transient response and stability, the PID controller takes advantage of three methods of compensation: proportional, lead, and lag compensation. Some desirable characteristics in the controller thus include high DC and low-frequency gain for good tracking performance and ample gain and phase margin for stability.

A proportional compensator is the simplest closed-loop system configuration and provides a way to move the open-loop magnitude response up and down to achieve a certain phase margin. The proportional compensator will generate a stable closed-loop system if the open-loop system is stable and the gain is sufficiently low. This is true because closed-loop poles approach open-loop poles for low gains [6]. For good tracking performance, however, the condition of low gain to maintain stability is too restrictive and thus more refined methods of compensation are needed.

The mechanical system has a typical bandwidth of tens or hundreds of hertz. Increasing the proportional gain can improve tracking performance, but creates

instability as a result of decreased phase margin. Introducing lead compensation can compensate for the lagging phase shift in the mechanical system at high frequencies. Lead compensation increases system bandwidth by adding positive phase shift in the region near crossover without significantly increasing the loop transmission magnitude in this region. A lead compensator is implemented as a proportional-plus-derivative (PD) compensator. The simplest form is a real zero of the form $\tau s+1$ (with a pole at infinity), but is more commonly implemented as the transfer function:

$$G_C(s) = \frac{1}{\alpha}\frac{\alpha\tau s + 1}{\tau s + 1} \tag{4.8}$$

which has a zero and a faster pole to the left for $\alpha>1$.

Having satisfied the condition of increasing phase margin to improve stability, the tracking performance can be improved by increasing the DC and low-frequency gain with a lag compensator. The effect of the lag compensator can be viewed in two ways. Lag compensation decreases high-frequency gain, which lowers the crossover frequency and increases phase margin for systems with monotonically decreasing phase and magnitude over frequency. Alternatively, the lag compensator may be seen to increase low-frequency gain while maintaining phase margin. A lag compensator has a transfer function of the form:

$$G_C(s) = a_0\frac{\tau s + 1}{\alpha\tau s + 1} \tag{4.9}$$

which has pole and a faster zero to the left for $\alpha>1$. In the extreme case where the pole is placed at zero, the lag compensator becomes:

$$G_C(s) = \frac{a_0(\tau s + 1)}{s} = a_0\tau + \frac{a_0}{s} \tag{4.10}$$

which is the transfer function of a proportional-plus-integral (PI) compensator.

Care must be taken when using the lag compensator in the placement of the lag zero to keep the negative phase dip well below the crossover point. Locating the lag zero too close to crossover can produce instability. Locating the lag zero at too low a frequency sacrifices performance by reducing gain over a wider frequency range [6].

A proportional-plus-integral-plus-derivative (PID) controller thus combines these three compensation techniques. The most basic transfer function of the PID controller is thus of the form:

$$G_C(s) = K_P + \frac{K_I}{s} + K_D s \tag{4.11}$$

where K_P, K_I, and K_D are the proportional, integral, and derivative gains, respectively.

4.3.2 Discrete-Time Implementation

The mechanical model of Equations 4.4 and 4.5 suggests that the controller should respond to both the position of the roller as well as its derivative, the velocity. This naturally suggests proportional-plus-derivative (PD) control. While PD control is commonly implemented using operational amplifiers and the like, a discrete-time controller is desired so that it can be implemented in a microcontroller. To make the controller more general, integral feedback was also added, but is not used by the present implementation of inTouch-3. Thus, implementation of the controller involves discretization of Equation 4.11.

The PID controller takes as inputs the desired position and the actual position. The encoder implicitly digitizes the position of the roller, which is represented by a digital counter, and this value is taken as the actual roller position. The desired, or reference, position is taken from the remote node through the serial interface. The difference between the desired position and the actual position is the error. The output of the controller is computed by multiplying the error by some constant and adding it to a multiple of the derivative of the error and the integral of the error. The derivative of the error is approximated in real-time as the first difference of the error. The integral is approximated using an error accumulator whose initial state is taken to be zero. The result is shown in Equation 4.12.

```
output[n] = KP * error[n] + KI * integ_error[n] + KD * deriv_error[n]          (4.12)

error[n] = desired[n] - actual[n]
integ_error[n] = integ_error[n-1] + error[n]
deriv_error[n] = error[n] – error[n-1]
```

where KP is the proportional gain, KI is integral gain, and KD is the derivative gain. To compensate for both mechanical friction and inductive friction, as described in Section 3.5.2, the actual velocity of the roller is used, approximated again by the first difference of actual positions. The output is thus computed as:

```
output[n] = KP * error[n] + KI * integ_error[n] + KD * deriv_error[n]          (4.13)
          + KA * (actual[n] – actual[n-1])
```

where KA is the friction compensation (anti-damping) gain. Note that Equation 4.13 is the computational analogy of the mechanical model, Equation 4.6, if the integrator is

removed. In the present implementation of inTouch-3, Equation 4.13 is recomputed at a sampling rate of 1kHz.

As a practical point, the system described by the difference equation above is subject to a condition known as integral wind-up. Since the integrator serves as an accumulator of error, a large integral term can develop, which can lead to excessive overshoot. Integral wind-up may occur, for instance, when the controller attempts to accelerate the mechanical system beyond its maximum rate, which produces an extended period of error. To prevent this condition, a limit may be imposed upon the integral term so that a large magnitude accumulated error does not pass into the output. Furthermore, output limits are imposed on the controller to protect the motor from excessive current.

4.3.3 Frequency-Domain Analysis

To aid in analysis of the inTouch-3 system in the next section, the PID controller will be examined in the frequency domain. Expansion of Equation 4.13 yields:

```
output[n] = KP * (desired[n] – actual[n])
          + KI * sum(0,n) (desired[n] – actual[n])
          + KD * (desired[n] – actual[n]-desired[n-1]+actual[n-1])
          + KA * (actual[n]-actual[n-1])
```
(4.14)

where `sum(0,n)` represents the sum from sample 0 to sample n of the error. The limits to the accumulator and the output are ignored in this analysis for simplicity. Taking the z-transform of Equation 4.14 gives:

$$\begin{aligned} Output(z) &= K_P(Desired(z) - Actual(z)) \\ &+ \frac{K_I}{1-z^{-1}}(Desired(z) - Actual(z)) \\ &+ K_D(Desired(z) - Actual(z) - z^{-1}Desired(z) + z^{-1}Actual(z)) \\ &+ K_A(Actual(z) - z^{-1}Actual(z)) \end{aligned} \tag{4.15}$$

which may be simplified to the following form:

$$Output(z) = H_{desired}(z)Desired(z) - H_{actual}(z)Actual(z) \tag{4.16}$$

where $H_{desired}(z)$ and $H_{actual}(z)$ are given as follows:

$$\begin{aligned} H_{desired}(z) &= (K_P + K_D) - K_D z^{-1} + \frac{K_I}{1-z^{-1}} \\ &= \frac{(K_P + K_I + K_D)z^2 - (K_P + 2K_D)z + K_D}{z(z-1)} \end{aligned} \tag{4.17}$$

$$H_{actual}(z) = (K_P + K_D - K_A) - (K_D - K_A)z^{-1} + \frac{K_I}{1 - z^{-1}}$$
$$= \frac{(K_P + K_I + K_D - K_A)z^2 + (2K_A - K_P - 2K_D)z + (K_D - K_A)}{z(z-1)} \quad (4.18)$$

Since integral feedback is not used in the present implementation of inTouch-3, the integral gain K_I can be set to zero. Equations 4.17 and 4.18 thus simplify to:

$$H_{desired}(z) = (K_P + K_D) - K_D z^{-1} = \frac{(K_P + K_D)z - K_D}{z} \quad (4.19)$$

$$H_{actual}(z) = (K_P + K_D - K_A) - (K_D - K_A)z^{-1} = \frac{(K_P + K_D - K_A)z - (K_D - K_A)}{z} \quad (4.20)$$

Using Equation 4.16, the PID controller located in the control block of Figure 4.1 may be represented by the following block diagram. Note that *Output(z)*, after discrete to continuous-time conversion, commands the servo amplifier, which produces I_m.

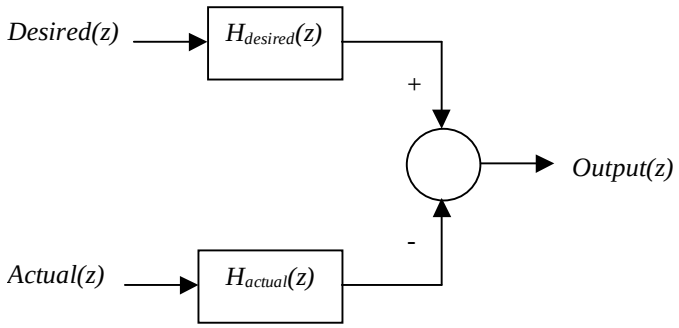


Figure 4.5: Block diagram of the PID controller

4.4 System Analysis Models

The inTouch control algorithm may be analyzed in four configurations: single-node zero disturbance torque, single-node zero reference input, two-node zero delay, and two-node non-zero delay models. These configurations will be examined in this section.

4.4.1 Single-Node Model with Zero Disturbance Torque

Under the single-node model, a node that is not connected to any other node is analyzed. In the standard two-node configuration of the inTouch-3, the remote encoder count is taken as the desired position and the local encoder count taken as the actual position. In this way, the two nodes are coupled together as expected.

In one configuration of the single-node model, the desired position serves as the input and the actual position as the output, Figure 4.6. The disturbance torque, T_d, is taken to be zero. The block between V_a and T_m represents the response of the servo amplifier and the motor. K_m is again the torque constant of the motor and K_{VI} is the transconductance of the Copley servo amplifier, which has a bandwidth of 3 kHz.

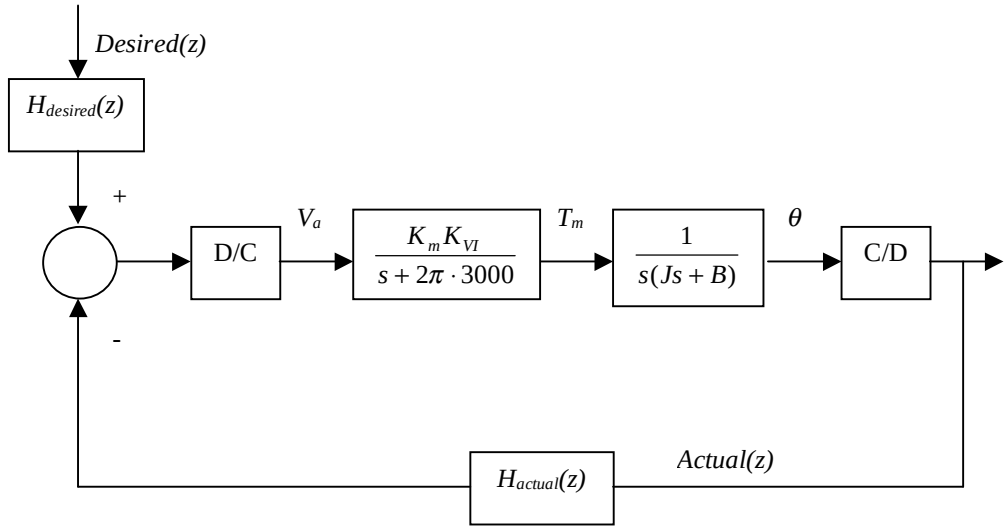


Figure 4.6: Single-node, zero disturbance torque model

The mechanical subsystem has a transfer function given by:

$$H_{mech}(s) = \frac{K_m K_{VI}}{s(Js + B)(s + 2\pi \cdot 3000)} \tag{4.21}$$

$H_{mech}(s)$ can be discretized to eliminate the D/C and C/D blocks by using the bilinear transformation, which maps the s-plane into the z-plane by the following relation:

$$s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} \tag{4.22}$$

where T is the sampling interval in seconds. For the analysis here, $T = 0.001$, representing a sampling rate of 1kHz. Applying Equation 4.22 produces:

$$H_{mech}(z) = \frac{K_m K_{VI} T^3 (z+1)^3}{4(z-1)[(BT + 2J)z + (BT - 2J)][(3000\pi T + 1)z + (3000\pi T - 1)]} \tag{4.23}$$

The open-loop transfer function for the feedback loop is $H_{mech}(z)H_{actual}(z)$. Applying Black's Formula and a series reduction yields the following for the system transfer function:

$$H_{sys}(z) = \frac{H_{desired}(z)H_{mech}(z)}{1 + H_{mech}(z)H_{actual}(z)} \tag{4.24}$$

The step response of this system is shown in Figure 4.7. The vertical axis represents the actual position of the mechanical roller. The mechanical constants and PID controller gains were chosen to produce representative behavior, but need to be determined empirically in future work under more thorough analysis. Note that the step response shows the performance characteristics described in Section 4.3.1: maximum overshoot, rise time, and setting time. The PID controller may be tuned to optimize the control system as measured by these criteria.

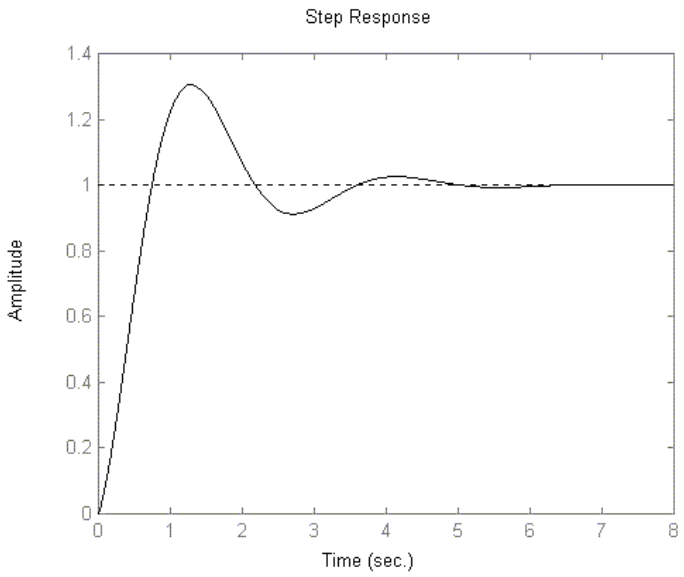


Figure 4.7: Step response of the single-node, zero disturbance torque model

4.4.2 Single-Node Model with Zero Reference Input

As an alternative configuration of the single-node model, the desired position can be taken as constant and zero with the disturbance torque serving as input and the actual position again taken as output, Figure 4.7. The K_{PID} block in the diagram is the PID controller of Figure 4.5.

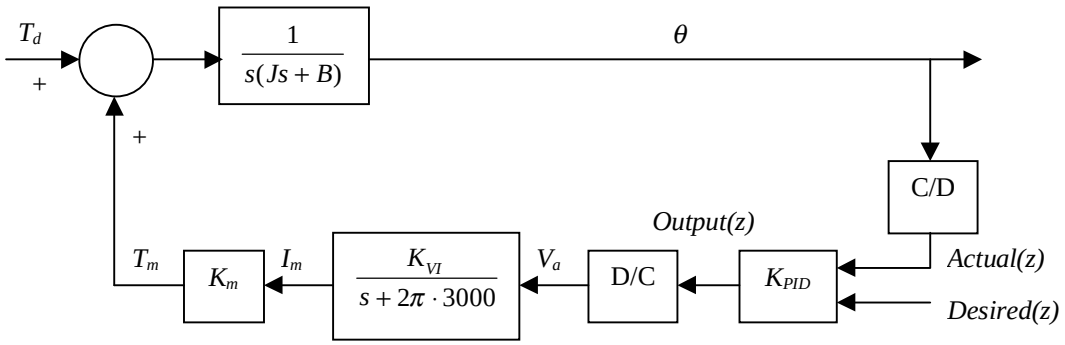


Figure 4.8: Single-node, zero reference input model

To analyze this model, the input *Desired(z)* will be taken to be zero. The K_{PID} block can then be reduced to:

$$Output(z) = -Actual(z)H_{actual}(z) \tag{4.25}$$

Using the bilinear transformation in terms of z, given by:

$$z = \frac{1+\frac{T}{2}s}{1-\frac{T}{2}s} \tag{4.26}$$

to convert $H_{actual}(z)$ to the continuous-time domain, yields:

$$H_{actual}(s) = \frac{(K_P + 2K_D - 2K_A)s + \frac{2K_P}{T}}{s+\frac{2}{T}} \tag{4.27}$$

The open-loop transfer function is then given by:

$$L(s) = -H_{actual}(s)\frac{K_m K_{VI}}{s(Js+B)(s+2\pi\cdot 3000)} \tag{4.28}$$

Black's formula can be applied to produce the system transfer function.

Given a constant and zero reference input, the controller will attempt to hold the mechanical roller in a fixed position. Since the input to the system is the disturbance torque, it is more useful to examine the impulse response rather than the step response. The impulse input represents a short perturbation of the mechanical roller from the zero position. The impulse response is shown in Figure 4.9. The vertical axis represents the mechanical roller position. The mechanical constants and PID controller gains were again chosen to produce a representative response. Note that the roller position settles to zero as expected.

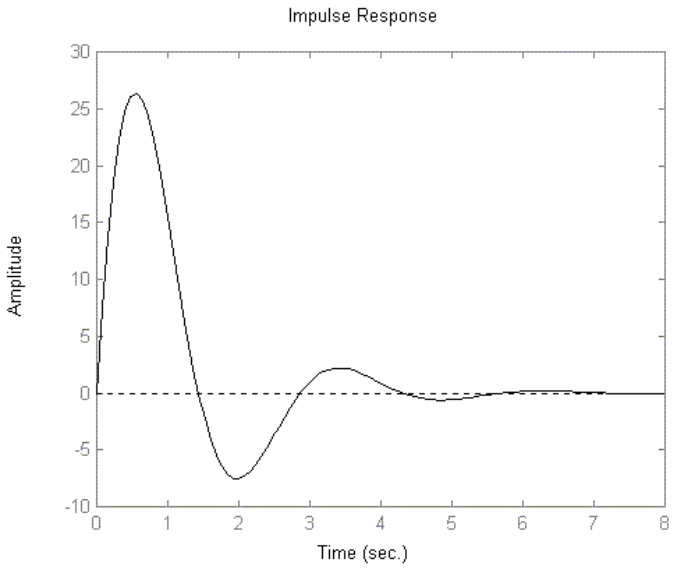


Figure 4.9 Impulse response of the single-node, zero reference input model

4.4.3 Zero Delay Model

A second system model is the two-node zero delay configuration. The two nodes are connected by an ideal, zero delay communications link. The interaction involves only one user, so the disturbance torque of one node is zero while the other is non-zero. In this case, the input is the disturbance torque, and the output is the actual position.

The full inTouch-3 system block diagram is shown in Figure 4.8. For the zero delay model, the D_{xx} blocks have zero sample delay.

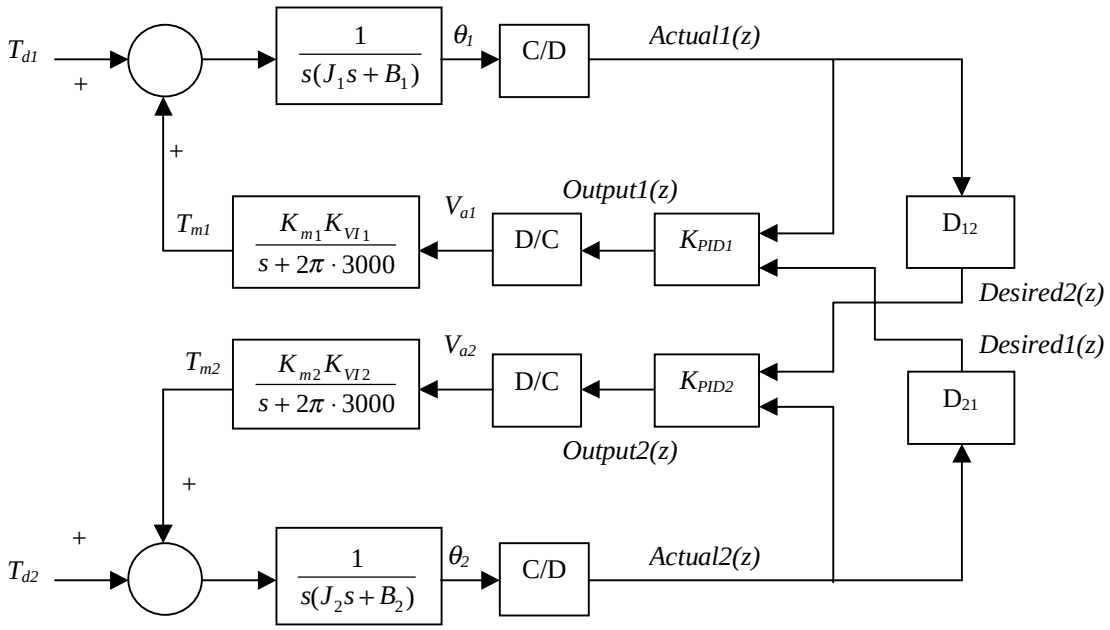


Figure 4.10: inTouch-3 system block diagram

In order to analyze the zero delay model, the full system block diagram can be reduced to a system of the form shown in Figure 4.9. The transfer function $H_L(s)$ represents the equivalent transfer function of the two PID controllers, the zero delay communications link and the mechanical subsystem of the remote node with zero disturbance torque.

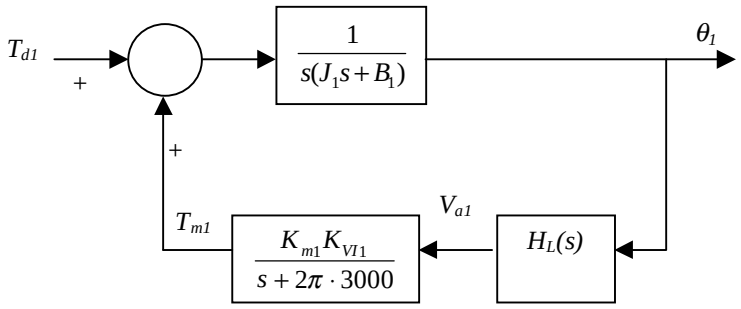


Figure 4.11: System diagram for analysis of the zero delay model

The first step in reducing Figure 4.8 to Figure 4.9 is to manipulate Figure 4.8 to account for zero delay and zero disturbance torque in the remote node. The result is shown in Figure 4.10.

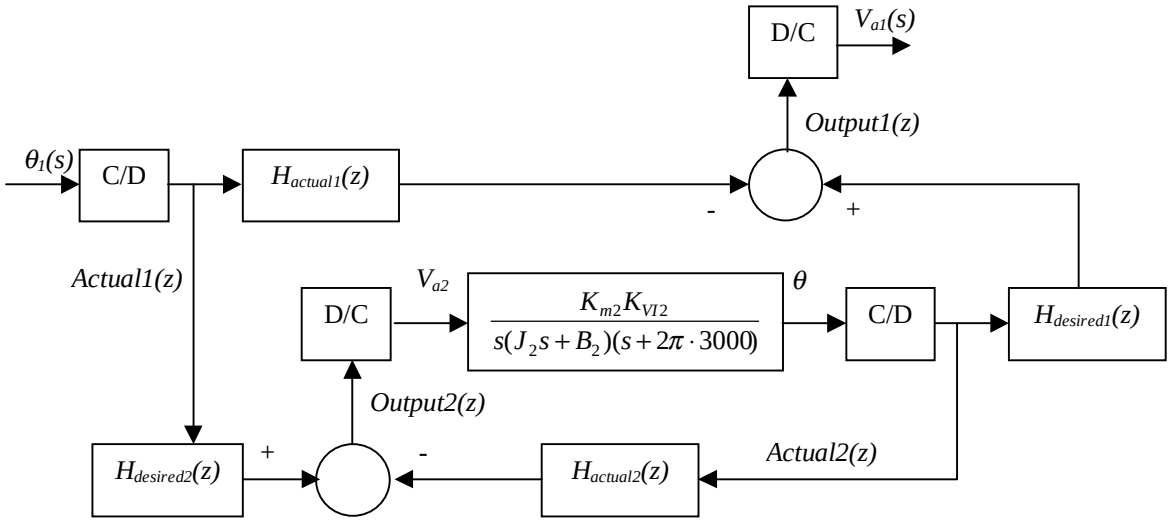


Figure 4.12: Block diagram of zero delay model

Since the disturbance torque for the remote node, T_{d2}, is zero, the mechanical subsystem and servo amplifier can be represented by the following transfer function:

$$H_{mech2}(s) = \frac{K_{m2}K_{VI2}}{s(J_2 s + B_2)(s + 2\pi \cdot 3000)} \tag{4.29}$$

$H_{mech2}(s)$ can be discretized to eliminate the D/C and C/D blocks by using the bilinear transformation, producing:

$$H_{mech2}(z) = \frac{K_{m2}K_{VI2}T^3(z+1)^3}{4(z-1)[(B_2T + 2J_2)z + (B_2T - 2J_2)][(3000\pi T + 1)z + (3000\pi T - 1)]} \tag{4.30}$$

The inner feedback loop consisting of $H_{mech2}(z)$ and $H_{actual2}(z)$ can be replaced by its closed-loop equivalent using Black's formula, yielding $H_I(z)$, given by:

$$H_I(z) = \frac{H_{mech2}(z)}{1 + H_{mech2}(z)H_{actual2}(z)} \tag{4.31}$$

Through series and parallel reductions, the block diagram of Figure 4.10 can be simplified to $H_L(z)$, given by:

$$H_L(z) = -H_{actual1}(z) + H_{desired2}(z)H_I(z)H_{desired1}(z) \tag{4.32}$$

$H_L(z)$ is the discrete-time equivalent of $H_L(s)$, which can be derived using the bilinear transformation. Substituting $H_L(s)$ into block diagram of Figure 4.9, the open-loop transfer function is found to be:

$$L(s) = \frac{K_{m1}K_{VI1}}{s(J_1 S + B_1)(s + 2\pi \cdot 3000)} H_L(s) \tag{4.33}$$

The impulse response of the system is shown in Figure 4.13, where the vertical axis represents the mechanical roller position of node 1. The mechanical constants and PID controller gains were chosen again to produce a representative response. As expected, the roller position settles to some displacement following the impulse disturbance torque. The value of this displacement depends primarily on the system damping. Less damping will obviously cause the roller to travel further before coming to rest. Note that the roller position does not return to zero as in the single-node, zero reference input model of the last section since node 2 is tracking the movements of node 1. The dynamics of node interaction will be examined in Section 4.5.

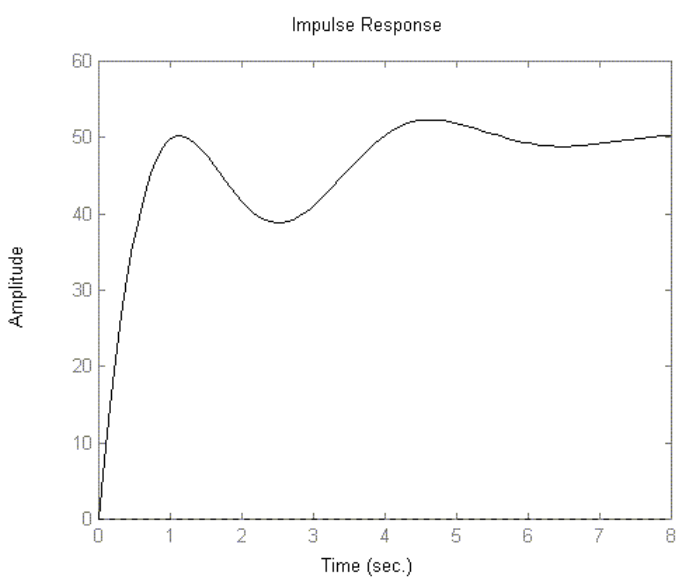


Figure 4.13: Impulse response of the two-node zero delay model

4.4.4 Non-Zero Delay Model

The third system model is equivalent in form to the zero delay model except non-zero delay is added to the communications link. In this case, the D_{xx} blocks of Figure 4.8 have non-zero sample delay.

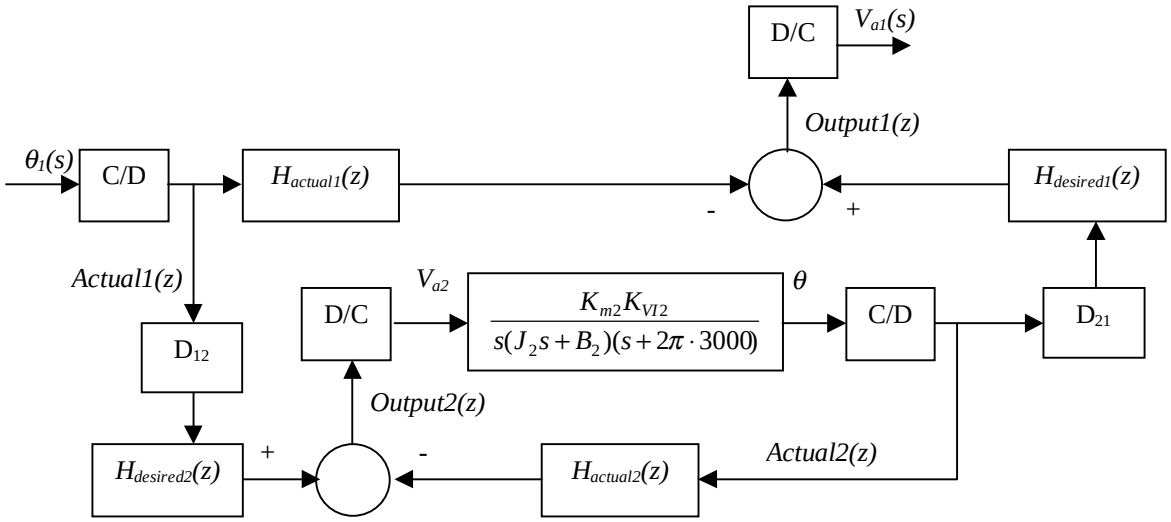


Figure 4.14: Block diagram of non-zero delay model

Figure 4.11 shows the block diagram for the non-zero delay model. The effect of the delay blocks is to produce a z^{-n} term as a result of the n-sample delay. The analysis here proceeds in an analogous manner to the previous section. The z^{-n} terms may be conveniently incorporated into $H_{desired}(z)$. Following the derivation of the last section, Figure 4.11 can be reduced to the following transfer function:

$$H_L(z) = -H_{actual1}(z) + z^{-n_{12}-n_{21}} H_{desired2}(z) H_I(z) H_{desired1}(z) \qquad (4.34)$$

where $z^{-n_{12}}$ corresponds to delay from node 1 to node 2 and $z^{-n_{21}}$ corresponds to delay from node 2 to node 1. The open-loop transfer function is then given by Equation 4.33.

The impulse response of the system is shown in Figure 4.15. By raising the gain and delay appropriately, an unstable response as illustrated in the figure can be attained. The instability seen was achieved after increasing the gain 50-fold and inserting a 16-sample delay. In practice, the inTouch system has much lower gain and phase margin and exhibits instability with delays above a few samples. The reasons for this discrepancy may include system poles not accounted for by the model as well as inaccurate mechanical constants. For future analysis, a more complete model could be used and the mechanical constants should determined precisely.

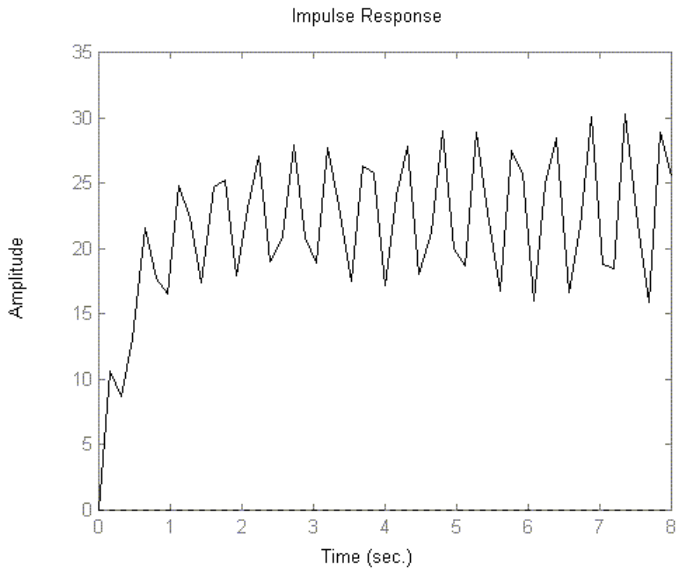


Figure 4.15: Impulse response of the two-node, non-zero delay model

4.5 Interaction Dynamics

Recall from the discussion of the PWM servo controller in Section 3.5.4 that good near-zero current control was necessary for the inTouch system. Now that the control algorithm has been developed, it can be used in the discussion of the interaction dynamics, which will lead to a justification of why this property is needed.

Consider for the moment single-user interaction with a node. As the user stimulus changes the node's local encoder counter, this information will be transmitted to the remote node. The remote node, which has no disturbance torque (i.e. no user stimulus), will track the local encoder changes. If it follows perfectly, then the difference in the positions, or the error, will be zero. Therefore, the calculated torque in the remote node will be close to zero. There will be small perturbations in the calculated torque since a small, but non-zero, amount of torque is needed to move the roller as it tracks the local roller position. However, these perturbations center around zero, so precise current control around zero is necessary for good tracking performance.

Consider now user stimulus at both nodes. The torque produced at each node will be a function of the difference in positions between them (recall the synchronization algorithm). It is unlikely that the difference in positions will differ by large amounts for any extended period of time. The reasons for this are that, first, users tend interact

with the device in a back-and-forth movement and, second, the restoring force generated by a large difference in position will be great and in the direction of decreasing difference. Therefore, good current control around zero is of paramount importance so that the tactile effect of inTouch is preserved and so that the controller can make the position difference converge to zero. Moreover, the tactile sensitivity of the hand will detect any imperfections in this control.

4.6 Adjustment of Algorithm Parameters

To facilitate adjustment of the algorithm parameters, two potentiometers were added to the controller board. Changing the resistances of the potentiometers by turning their adjustment screws changes the value of the analog voltage presented to the PIC microcontroller. Using the microcontroller's internal ADC, the analog voltage can be mapped to digital values used to set the algorithm parameters. In this way, the algorithm's parameters can be adjusted easily without having to reprogram the microcontroller. Moreover, since the parameters may be adjusted while the algorithm is executing, immediate and dynamic feedback about the system's response can be obtained.

The controller board and firmware also allow these digital values to be set exactly. An example where this might be useful is the case where the algorithm is being tuned for the first time. It may be desirable to set values that are initially likely to work based on previous experience with other units. In order to see what the digital values are, an adjustment mode was added to the firmware, which is entered by setting a jumper on the controller board. In adjustment mode, the algorithm parameters are sent to the serial interface where they may be viewed on any serial terminal. The output also includes other information such as the firmware revision and state of other jumper-configurable settings.

The digital values produced by the ADC may be assigned to any of the algorithm's parameters merely by changing the microcontroller code. It has been found that the two most useful parameters that may be need to be adjusted are the derivative gain in the PID filter and the anti-damping gain. Adjustment of these parameters and how they influence the system's response will be outlined here.

The derivative gain controls the amount of damping in the system. The parameter is adjusted by perturbing the rollers slightly with the node in isolation, that is, disconnected from the remote node. Since the remote position is unchanged while the node is disconnected, it may be seen from the algorithm implementation above that the roller will oscillate back and forth as it attempts to maintain the same position. By adjusting the derivative gain while perturbing the rollers, a point where the oscillation decays quickly can be achieved, indicating a critically-damped system.

The anti-damping gain decreases the perceived friction of the rollers by adding a small amount of velocity-dependent excitation to overcome mechanical and inductive friction. By increasing the anti-damping gain, a point can be achieved where the perceived friction is reduced to negligible levels and the roller appears to spin freely. Note, however, that the efficacy of this type of compensation is greatly dependent upon the uniformity of mechanical friction as the roller rotates about its axle. If the friction is highly variable, then the velocity information obtained by computing the difference in encoder values will have substantial error, and the effect of the compensation will be diminished.

Looking again at the algorithm, it may be seen that the best method of adjusting the anti-damping gain is the state where there is no difference between the local and remote positions. This condition can be obtained by placing the device in loopback. In loopback, the transmitted encoder information is routed immediately back to the device. This simulates the condition of zero error; the local and remote positions will always be the same, thus canceling all but the last term in Equation 4.13. What will remain is the anti-damping term. By adjusting the gain to a sufficient value, a low friction state can be obtained.

5 Production

Twenty inTouch units comprising ten sets were ultimately made for internal use and distribution to corporate sponsors. The production effort involved selection and acquisition of materials and services, outsourcing of labor-intensive tasks, and in-house assembly, testing, and shipment of units.

5.1 PCB Layout and Manufacturing

The first step in producing the inTouch-3 system was the layout of the controller and auxiliary printed circuit boards (PCB). For this purpose, Protel Client software was used. A two layer (top and bottom) PCB format was selected for each circuit for simplicity and lower cost.

The process of generating a PCB design is first drawing the schematic in Client's schematic editor. The schematic can then be used to generate a file known as a netlist, which contains all the components of the circuit, their PCB footprints, and their electrical interconnections. The netlist allows the PCB editor to guide the user in routing the components. Routing involves drawing traces on the PCB to electrically connect components. Client also provides the option of auto-routing using many configurable optimization algorithms. The inTouch controller board and auxiliary board in the mechanical unit were routed manually since they are relatively simple designs.

The controller board was designed to connect to the Copley servo amplifier by means of interconnect pins on the Copley unit. In this way, the whole controller unit could be assembled simply by aligning the interconnect pins and fastening the controller board to the servo amplifier with aluminum stand-offs. Therefore, the Copley servo amplifier defined the mechanical outline of the controller board. The controller board and the assembled controller unit are shown below. The PIC microcontroller plugs into the IC socket in Figure 5.1.

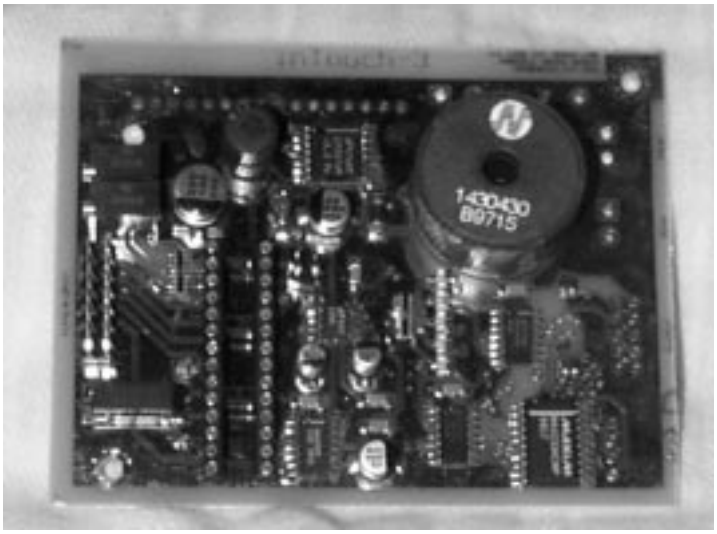


Figure 5.1: Controller board

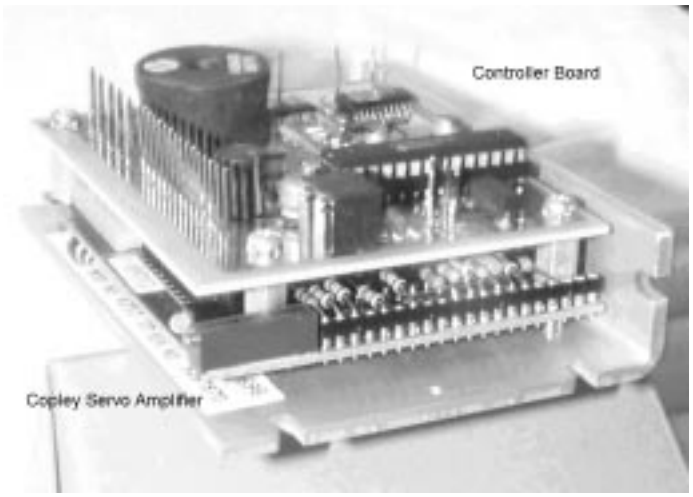


Figure 5.2: Assembled controller unit

To achieve the small board size necessary, surface mount form factor versions were selected for each component where possible. Materials acquisition proved to be the most time-consuming task as several parts were backordered for several weeks. Components with second-source manufacturers should be selected whenever possible to prevent delays caused by component shortages.

After the components were placed and routed, the top layer of the controller board was covered with a ground plane. The ground plane is a large area of copper connected to ground that covers the area of the board not containing components or traces. The use of a ground plane lowers ground lead inductance, which reduces electrical noise, particularly for the analog control signals on the board, and helps control ground bounce. Ground bounce occurs when there are large currents flowing through the ground electrode caused by high-speed switching logic, for example. This flow of charge can induce a voltage in the ground lead if ground lead inductance is sufficiently high. The result will be fluctuations in the ground potential which may cause glitch sensitive logic, whose levels are referenced to ground, to trigger mistakenly. Moreover, if the ground potential varies, then the analog control voltages to the servo amplifier may be adversely affected.

The second circuit to lay out was the auxiliary board, which implements the auxiliary circuit described in Section 3.10. The auxiliary board was designed to fit snugly in the base of the mechanical unit. Connectors were selected so that readily available

connecting cables could be used between the auxiliary board and the control box. The auxiliary board also connects to the rotary encoder of each motor by means of flex cable. The auxiliary board by itself and mounted to the mechanical unit are shown below.

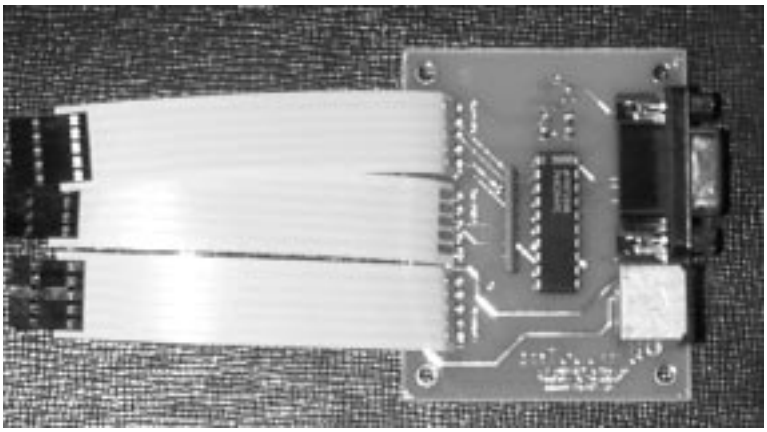


Figure 5.3: Auxiliary board

Figure 5.4: Auxiliary board mounted to the mechanical unit

After the printed circuit boards were laid out and finalized, they were output to an industry standard format known as Gerber format. Gerber files are used by computer-controlled machinery to manufacture the printed circuit boards. The actual PCB manufacturing was carried out by Alberta Printed Circuits, who have a short turn-around service of two business days.

5.2 PCB Assembly

Assembling printed circuit boards can be an extremely tedious task. This was especially true considering that each inTouch unit required three controller boards for each of the twenty units that were to be manufactured. Therefore, Best Technologies, Inc. was selected to assemble the boards to specification at a nominal cost. The components and printed circuit boards were then shipped to Best Technologies for assembly.

5.3 Control Box

With the controller units completed, the next issue to resolve was that of packaging. Recall that the proposed scheme was to enclose the electronics in a control box located some distance away from the mechanical unit. An aluminum box seemed an appropriate choice for the control box, given that it was easy to machine and was available in many different sizes. It was soon realized that while the task of cutting holes for connectors was feasible for one or two boxes, it would not be so easy for twenty. The final machining was outsourced to a local machine shop, Eastern Tool Corporation, who also fabricated the metal parts for the mechanical unit.

Midway through the production process, it was discovered that the initial power supply was insufficient to absorb the transient currents generated by the motors, as detailed in Section 3.9. Thus, a power supply larger in both capacity and size was selected, but which no longer fit in the control box. New aluminum boxes were selected and prepared, but additional cost was obviously incurred in the change.

To handle the problem of securing the controller boards and power supply to the control box, screw holes were drilled in the initial aluminum box. In this way, the controller boards and the power supply could be fastened to the sides of the box with the screws extending inward from the outside. After the change in aluminum boxes, it was decided that a simpler method of mounting was to use adhesive foam, which not only secured the boards to the sides of the box, but also isolated them electrically from each other and the power supply.

5.4 Cabling

The mechanical unit is connected to the control box by means of two cables. The encoder cable is a standard, 9-conductor DB9M cable commonly used as an extension cable for personal computers. The motor power cable uses mini-DIN8 connectors similar to those found on the serial ports of Macintosh computers. After some preliminary tests, however, it was decided that a larger gauge cable should be used since the motor power cable might be required to carry moderately large currents. In the worst case, the cable would need to carry 3.75 amperes, since all three motors are powered from the cable and recall that each has a current limit of 1.25 amperes. Therefore, custom cables were ordered from Communications Supply, Inc.

The two control boxes in each set of inTouch-3 systems are connected to one another by means of an 8-conductor flat modular cable that carries serial data between controller units. Recall that each roller requires one controller unit and each roller operates independently of the other two in each node. Therefore, three serial communication channels are required to connect the control boxes. At minimum, each serial channel requires signal ground, transmit data (TD), and receive data (RD) lines. Since only eight conductors were available, only two signal grounds were included. The control boxes are equipped with DB9M connectors for the communications port. An RJ-45 to DB9M adapter was included to make the connection. The advantages of flat modular cable over thicker twisted serial cable are cost and ease in changing the cable length.

Finally, each control box receives AC power through a 3-conductor cable connected to the mains. The system connections of inTouch-3 are summarized in the figure below.

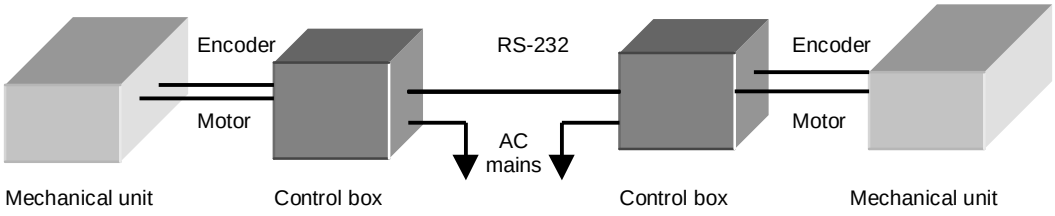


Figure 5.5: inTouch-3 system connections

5.5 Assembly and Testing

Two students were hired to assist in the assembly. Their work consisted of mounting control units and connectors, making the necessary solder connections, and finally testing the assembled units.

As a result of tolerances in the wood components of the mechanical unit, each was hand-adjusted for optimum performance. These imperfections were primarily a result of the hand-machining of these components. Perhaps in future versions, the mechanical design can be adjusted to tolerate these imperfections, or the wood parts may be replaced completely.

Moreover, several of the controller boards assembled by Best Technologies possessed slight defects, which were uncovered during testing. The primary cause of these defects was shorts to the ground plane. These problems could have been avoided if solder masking was used. Solder masking covers the traces on the printed circuit board to prevent shorts to other circuit traces and the ground plane. Solder masking was not used primarily because of time constraints; the process of masking can add several weeks to the PCB production time. As a result of this design choice, the ground plane, which covers the areas around the components was left unmasked, and can easily short to the traces if careful soldering is not observed. Most of these defective boards were later corrected by hand.

6 Modem Interface Unit

6.1 Overview

Work was started to develop an external modem interface unit for the inTouch -3 system to permit operation over long distances through the telephone network. The fact that a standard RS-232 serial interface was used for data communication between control boxes permits this transition with little or no change to the data format. However, the issues of timing and communications latency are raised, which will be examined here first before proceeding into the implementation of the interface unit.

6.2 Timing and Delay

Recall from Chapter 5 that the serial connection of inTouch-3 contains three independent serial channels that are not synchronized. What this means is that if a modem is to be used as the communication link, a method of multiplexing or synchronization will be required. Multiplexing will be discussed in this section and synchronization will be explored in Chapter 8.

Delay must first be addressed. Previous experience has shown that the inTouch system is very sensitive to delay on the order of a few milliseconds. Any type of multiplexing scheme will introduce some amount of delay into the system. The average service time will be one half of the expected service time of each channel multiplied by the number of multiplexed channels for a time-division multiplexed (TDM) system. Synchronizing schemes are advantageous here in that they introduce less delay. The time spent waiting in queue in a TDM system is instead distributed over the sampling interval in a synchronous system. This will become apparent in the discussion of Chapter 8.

While the delays introduced by multiplexing or synchronizing schemes are easily accounted for, network delays are not so readily determined. In a typical data network that does not have guaranteed service, such as the local area network used in inTouch-2, delay is neither uniform nor bounded. Fortunately, voice networks are required to carry real-time data, where lost or delayed packets are useless. To achieve real-time performance, voice networks tend to operate isosynchronously, dividing link capacity equally between channels. This provides for uniform delay between packets since in this scheme it is not possible for one channel to load the network in such a way as to deny service to other channels. Moreover, delay is bounded since normal conversations are difficult over high-latency channels without more extensive conversation protocols such as those used by radio operators.

Voice networks are thus suitable for the inTouch-3, which is also a real-time system. Furthermore, since delay is more predictable, compensation techniques can be simpler.

6.3 Hardware Implementation

The hardware implementation of a serial multiplexer/demultiplexer will be explored first. This device combines multiple streams of data into one and separates the

multiplexed stream into its parts again on the remote side. In the multiplexing unit, the streams of data arrives at *channels* and are combined into an outgoing stream of data sent to the modem, referred to as the *link*. In the demultiplexing unit, data arrives from the link and is separated and retransmitted on the channels. Note that the system is full-duplex; data travels in both directions simultaneously on both channels and the link, as shown in the figure below. To operate with the inTouch-3 unit, which contains three independent serial channels, a 3-to-1 multiplexer/demultiplexer is thus required.

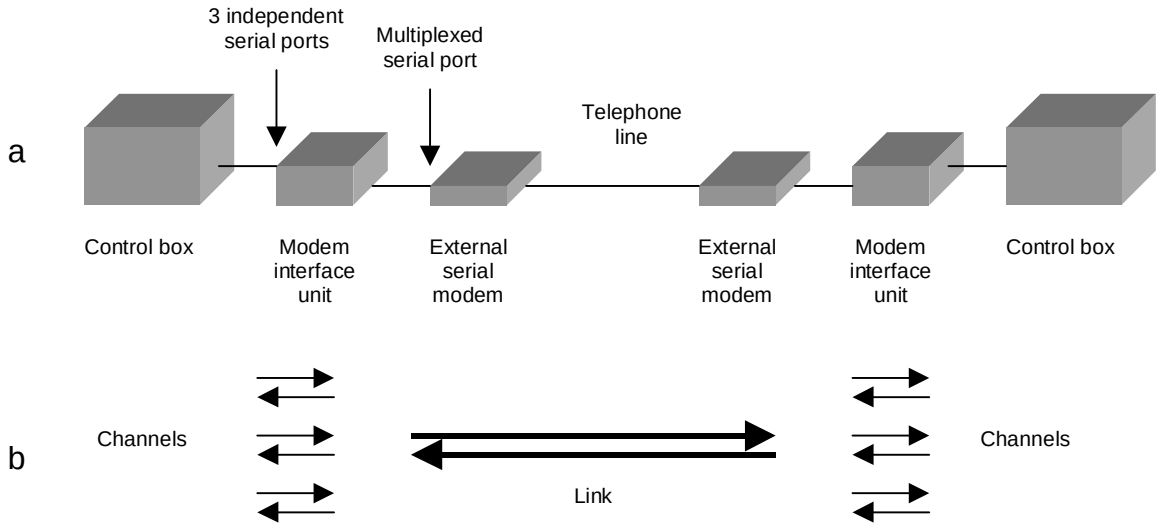


Figure 6.1: Serial multiplexer/demultiplexer system connections (a) and data flow (b).

6.3.1 Serial Ports

A straightforward way of implementing this device is to build a microcontroller circuit with four serial ports. Most microcontrollers with internal serial ports such as the PIC 16C73A contain at most only two serial ports. However, the Maxim MAX3100 provides the answer. The MAX3100 is an external universal asynchronous receiver transmitter (UART), which handles transmission and reception of RS-232 data. The MAX3100's interface to the microcontroller is via SPI, described in Section 3.6. SPI is a bus interface that permits multiple devices to share the same set of communication lines, Figure 6.2.

Note that data out (SDO) from the microcontroller connects to data in (SDI) of the device and vice versa. A device on the bus is selected by means of its chip select (CS) pin, which activates the chip and tells it to listen and respond to data on the bus. Thus, a

microcontroller can control any number of devices on the SPI bus, essentially limited only by the number of chip select outputs.

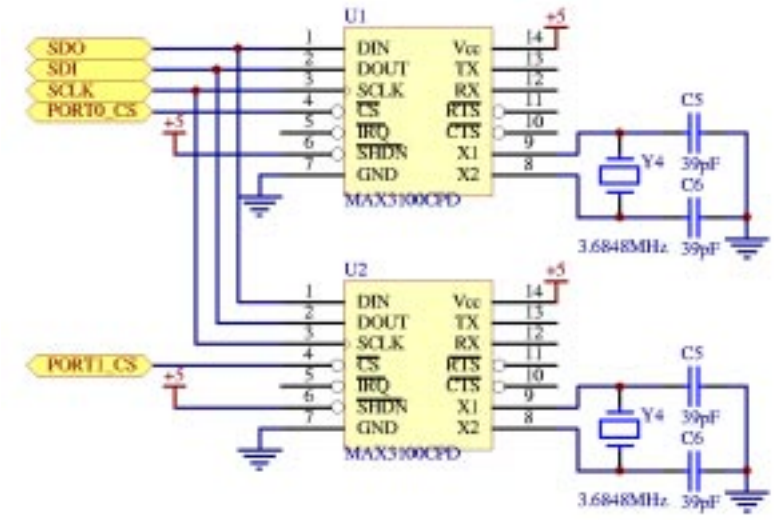


Figure 6.2: Multiple external serial ports

The MAX3100 UART expects logic level inputs and outputs at its TX and RX pins. In order to use it with RS-232 data, a level converter similar to the MAX233 must be used. While the MAX233 needs no external components, it only contains two receivers and transmitters. A four port serial multiplexer would thus require a minimum of two of these chips. The MAX208, which contains four receivers and transmitters, was selected to provide a more cost effective solution, as only one of these devices is needed. [1] The MAX208 does, however, require several 0.1µF external capacitors for proper operation of the internal charge pump, Figure 6.3.

[1] Two devices will actually be needed to implement control lines. This will be discussed in Section 6.4.1.

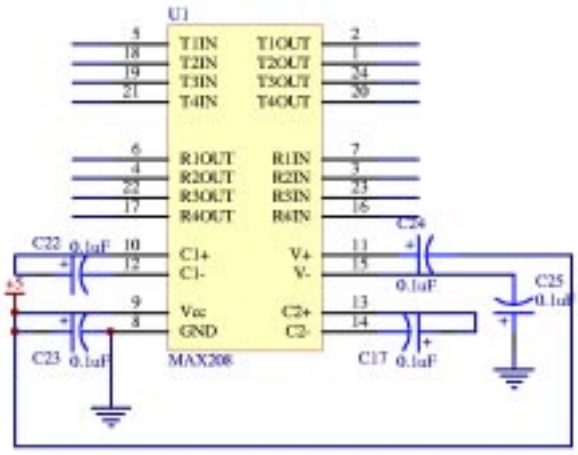


Figure 6.3: MAX208 four channel RS-232 level converter

6.3.2 LCD Display

In addition to four serial ports, a means of display was desired to show the state of the device, since it would also presumably handle session initiation and termination. A simple, low-power display device is the character liquid crystal display (LCD). Character LCD's commonly use the Hitachi HD44780 as their display controller IC. The HD44780, usually built into the LCD module, provides a straightforward command-oriented interface to the microcontroller, and requires either a 4-bit or 8-bit parallel interface for data and commands as well as several control lines.

In order to minimize usage of input/output pins on the microcontroller, which tend to be a precious commodity, the LCD was interfaced to the SPI bus using a serial-input-parallel-output shift register, the 74HC595, to convert serial data to parallel data. The shift register is clocked using the same SPI bus clock and receives its data from the SPI data output line. With this scheme and appropriate firmware, the microcontroller can address the LCD as though it were an SPI device, Figure 6.4.

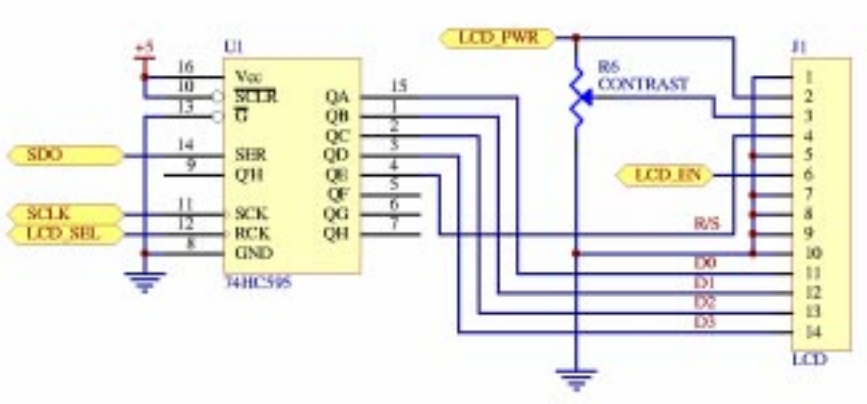


Figure 6.4: SPI-interfaced character LCD

6.3.3 Numeric Keypad

To complete the user interface, a means of input is needed. A numeric keypad provides the solution here. A standard 3x4 keypad, which has an arrangement similar to the keypad on a touch-tone telephone, was used to allow the user to input commands, and most importantly, the number to be dialed. There are many schemes to read keypads, which consist of a matrix of switch contacts. A simple and straightforward scheme is as follows. When a key is depressed, it connects one row contact with one column contact. If the microcontroller places a logical high or low on a row and examines each of the columns for the corresponding high or low level, then it can detect which key is pressed by repeating the process with each row.

Again to minimize usage of input/output pins, a method of making the keypad behave as an SPI device was developed, Figure 6.5. Using the 74HC595 shift register again, logical high or low levels can be set on the rows of the keypad by writing appropriate bytes to the shift register. The columns may be read with a parallel-input-serial-output shift register such as the 74HC165. Both the 74HC595 and 74HC165 can operate on the SPI bus. For simplicity in the final implementation, the 74HC165 was omitted and the columns were read from the microcontroller's input/output pins directly.

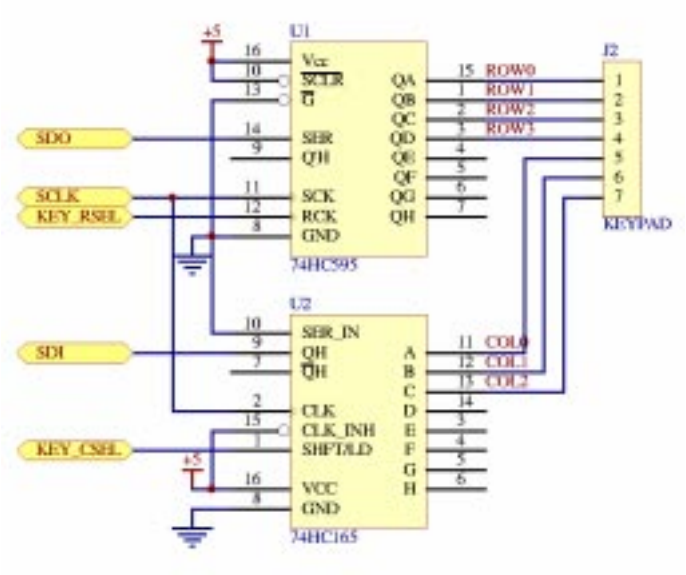


Figure 6.5: SPI-interfaced keypad

6.3.4 SPI Bus Address Decoder

A convenient device that can be used to address SPI bus devices is a decoder, Figure 6.6. The 74HC138 is a 3-to-8 decoder, which takes as input 3 bits and selects one of 8 control lines from the 3-bit binary number. Therefore, addressing a device on the SPI bus involves outputting a 3-bit number to select the appropriate device and subsequently reading or writing to the SPI bus.

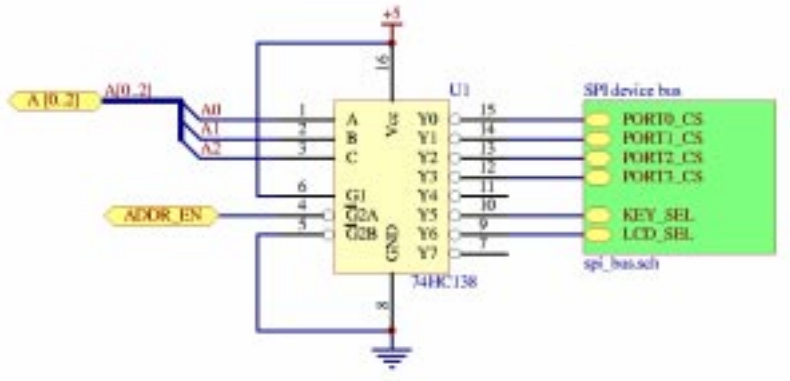


Figure 6.6: SPI bus address decoder

6.4 Firmware Implementation

Turning now to the microcontroller firmware, three primary functions can be identified. The microcontroller should communicate with the modem through one of the serial ports with the ability to sense when a connection has been established through the carrier detect line of the modem. Secondly, the microcontroller should read input from the user through the keypad and display both input and connection status on the LCD display. Finally, the microcontroller should handle the multiplexing/demultiplexing functions once the connection has been established. It should receive data from the three serial ports in a round-robin fashion and re-transmit it through the fourth serial port to the modem. At the same time, it should receive data from the modem and route it to each of other three serial ports again in round-robin fashion.

6.4.1 Modem Controller

The first of these tasks can be implemented quite simply as a result of the modem's standard AT command set and status lines. The modem may be thought of as a simple serial link with two modes, command and data. In command mode, the modem responds to AT commands received from the serial port, which include dialing, disconnecting, and so on. Once a connection has been established, the modem proceeds into data mode and acts as a simple serial communication channel.

The modem's interface to the microcontroller requires a full serial port as defined in the RS-232 standard. In addition to the transmit data (TD) and receive data (RD) lines, the serial port must also implement data terminal ready (DTR), ring indicator (RI), carrier detect (DCD), clear to send (CTS), and request to send (RTS) lines. In command mode, the modem only responds commands when DTR is asserted. In data mode, the modem only maintains the connection as long as DTR is asserted. RI indicates when an incoming call occurs so that the microcontroller can take appropriate action to answer. DCD indicates when a connection has been established with the remote modem. RTS governs flow control for data from the modem to the microcontroller. CTS governs flow control for data from the microcontroller to the modem. The action of the modem controller can therefore be summarized by the following tasks:

1. Issue a proper initialization string with the AT command.
2. Dial a number (obtained from the user interface module) using ATD.
3. Monitor RI for incoming calls and issue ATA to answer.

4. Monitor DCD after dialing or answering to know when to enable the serial multiplexer/demultiplexer.
5. Monitor DCD while connected in case the connection is lost.
6. Assert RTS to tell the modem the microcontroller is ready to accept incoming data for demultiplexing.
7. Monitor CTS and pause multiplexing as necessary if the microcontroller is exceeding the modem's channel capacity.
8. Deassert DTR to disconnect the call.

6.4.2 Keypad Input Parser and Display

The LCD display and the numeric keypad comprise the user interface to the device. The LCD, as the output device, shows the number to be dialed as it is entered on the keypad as well as the status of the connection. The keypad input parser stores the entered number in a queue until the user requests a connection to be established. The parser need only recognize a few commands, which are distinguished by the pound (#) key as a prefix. Pressing the pound key twice (##) dials the number stored in the queue and waits for a connection. Pressing pound and the star key (#*) aborts the current operation or hangs up the modem if connected and returns to ready state. The parser recognizes two other sequences that have special significance, (#1) and (#2). The meaning of these commands will be explained shortly.

6.4.3 Connection Types

The modem interface unit was designed to support three connection types: telephone network, direct modem, and null modem. A telephone network connection is established by two modems over the telephone network, and is the intended configuration for use with the inTouch-3. A direct modem connection involves wiring the modems together directly. In this case, no dial tone can be generated, but a connection can be established, as will be illustrated shortly. This configuration is most commonly used for demonstration purposes. The last type of connection, null modem, involves a direct cable connection between modem interface units, with no actual external modem present. A null modem is an adapter that is meant to connect two serial terminals together.

A telephone network connection is initiated by entering the number to be dialed on the keypad and pressing (##). No action is necessary on the remote end as the telephone

line will ring and the remote modem interface unit will answer the call. Once the connection has been established, the serial multiplexer/demultiplexer will be started automatically on each side.

A direct modem connection is initiated by pressing (##) on one side and (#2) on the other side. Recall that (##) dials a number with the "ATD" modem command. If no number is present in the queue, only the "ATD" string will be sent to the modem. The sequence (#1) is the manual answer command, which will send the string "ATA" to the modem, causing it to go off hook. The effect of this sequence will be a manual connection sequence initiated by the pair of modems, which solves the problem caused by the absence of a ring signal.

Finally, a null modem connection is initiated by entering (#2) into the keypad. The (#2) sequence starts the serial multiplexer/demultiplexer immediately without a connection sequence, since the null modem connection does not require one.

6.4.4 Data Routing

The serial multiplexer/demultiplexer performs time-division multiplexing (TDM) of the data stream. In time-division multiplexing, all channels share the capacity of the link equally by transmitting in predetermined time slots. As with any communications scheme, the issues of framing and routing must be considered. Framing and routing information is usually added to data packets in the form of header bytes to assist the receiver in distinguishing them. Since the link capacity of the modem interface unit is already restricted to the relatively low bandwidth of the modem, the amount of overhead in framing and routing information should be minimized for maximum transfer efficiency.

As another design consideration, the RS-232 protocol restricts packet size to byte multiples for the 8N1 format. In other words, any packet of length less than a byte will still require a full byte. Thus, while only two bits are needed to distinguish data from the three serial channels of the inTouch, a full header byte would be required. The resulting efficiency would be 50 percent (one header byte for every data byte), thus reducing the effective bandwidth by one-half.

A more efficient scheme is to use a TDM system where data is transmitted from each channel in a consistent order. For example, the first byte might be data from channel

1, the second byte data from channel 2, and the third byte data from channel 3. The sequence might be repeated four times and terminated by a framing byte, a reserved byte that cannot appear in the data. The framing byte delineates the data frame, which consists of the 12 data bytes plus the framing byte. The purpose of the framing byte is to resynchronize the receiver so that if data is lost in the frame, only a maximum of 12 bytes will be misrouted. Once the receiver sees the framing byte, it will route the next received byte to channel 1. This system is used in the present implementation of the serial multiplexer/demultiplexer, and has an efficiency of about 92 percent, since only one byte out of thirteen is overhead.

Another issue that must be considered is that of queuing. Since several streams of data are converging into one, the resulting link must have a capacity at least as great as total arrival rate of all channels. If the link capacity does not satisfy this constraint, then data queues will be needed to store data until it can be serviced. Care must be taken so that the arrival rate of all channels does not exceed the link capacity in such a way as to fill up the queues. If this occurs, the oldest or newest data can be discarded as appropriate according to the chosen service rule. In the present implementation of the modem interface unit, queues are implemented in hardware as a feature of the MAX3100 UART. The MAX3100 contains an eight byte receive queue, which is cleared if an overflow occurs. This condition should not occur as long as the link can maintain a speed of 57600 baud. This number is derived by considering that three 19200 baud channels are being multiplexed.

6.5 Test Results

With the hardware and firmware implementation of the serial multiplexer/demultiplexer complete, the entire circuit was laid out in Protel Client, and prototype printed circuit boards were ordered from Alberta Printed Circuits. The modem interface board is shown in Figure 6.7.

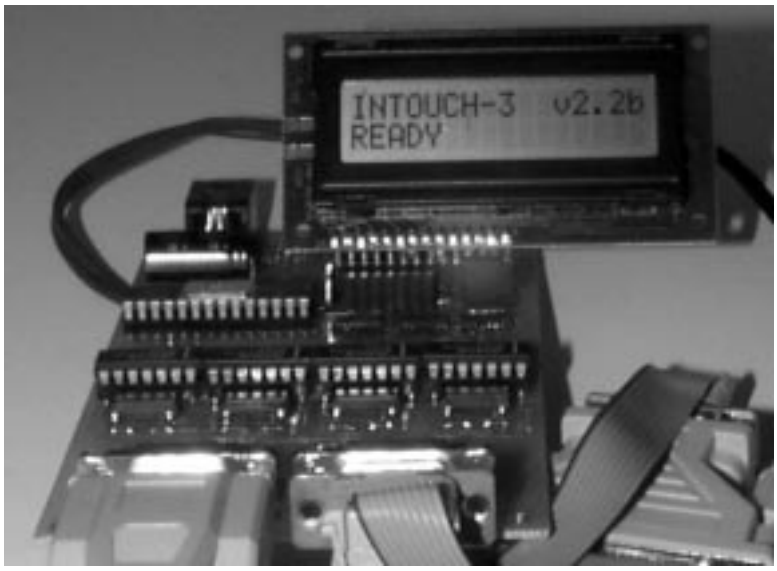


Figure 6.7: Modem interface board

The modem interface unit was tested with the inTouch-3 and two U.S. Robotics 56k modems. Although the modems can achieve a maximum connection speed of 53 kilobaud, this speed can only be attained in one direction and only if one side uses a digital phone line. With connections made between two standard analog lines, a maximum throughput of 33600 baud without compression can be achieved. This bandwidth ultimately proved to be insufficient for the present implementation of the inTouch-3 system. The reader is referred to the ITU V.90 standard for details of the 56k protocol.

Test results of the modem interface unit showed that the modem could not maintain the 57600 baud throughput required for inTouch-3, as expected. The observant reader might point out that the ideal maximum throughput of 115200 baud might be achieved with compression. However, the use of data compression in the modems presented unexpected problems of delay. The compression algorithm used in the U.S. Robotics modems were found to be not strictly real-time, and thus added variable communication latency to the system that is function of data entropy. This is expected since data compression is achieved by transmitting groups of bytes as shorter representations. Byte-to-byte compression would not be very effective since the algorithm could not take advantage of patterns in the data.

One solution to the modem bandwidth problem is to reduce the sampling rate of the inTouch-3 so that lower bandwidth is required. In changing the sampling rate, the

algorithm parameters will most likely need to be re-adjusted to stabilize the system. Another solution is to synchronize sampling between controller boards, so that multiplexing is unnecessary. In this case, only the modem controller and user interface would be required. Synchronization will be explored in Chapter 8.

7 Other Accessories

7.1 Port Adapter

To facilitate adjustment of the algorithm parameters as described in Section 4.6 and to view informational output on the serial terminal, a port adapter was designed and manufactured. The port adapter, shown in Figure 7.1, plugs into the communications port of the control box and permits easy access to the three independent serial ports. The RJ-45 to DB9M adapter used in the communications cable serves the dual purpose of connecting the port adapter to the computer's serial port. The pin allocation of the DB9M adapter is compatible with the port adapter. Thus, a standard, crossed 4-conductor flat modular cable with RJ-11 connectors, commonly used in telephone equipment, is all that is required.

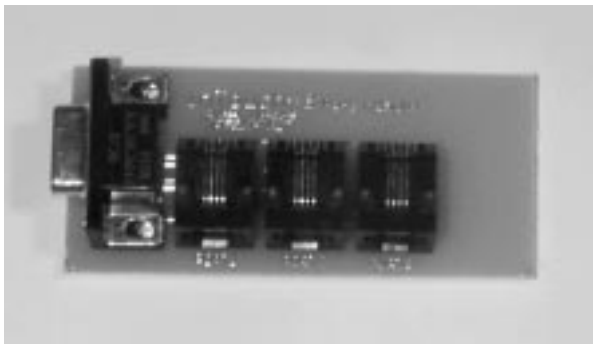


Figure 7.1: Port adapter

7.2 Delay Simulator

To simulate the effects of communication delay, a simple circuit was designed and implemented. The circuit is comprised of two PIC 16C73A microcontrollers and one MAX233 RS-232 level converter. Each microcontroller implements a size-adjustable FIFO buffer for data travelling in one direction. Serial data is received and

retransmitted after a sample delay set by the size of the buffer. The random-access memory capacity of the microcontroller is 192 bytes. A maximum buffer size of 128 bytes is achievable with the remaining memory consumed by program variables.

Using the same scheme developed for algorithm parameter adjustment in the inTouch, a potentiometer was interfaced to the internal ADC. The potentiometer allows for adjustment of the buffer's size without reprogramming the microcontroller. The serial output is used along with a standard serial terminal to display the buffer size as set by the potentiometer.

The contents of the buffer are advanced by the arrival of new data. Therefore, it is imperative that streaming data arrive at uniform intervals if consistent, predictable delay is desired. The current implementation of inTouch-3 only transmits data on changes in the encoder, which presents a problem under this scheme. There are two options available. The inTouch code can be altered to transmit the encoder counter at every sampling interval, regardless of whether the position has changed or not. Alternatively, the delay simulator can be connected to the synchronous clock output of the controller board, which is coherent with the sampling interval. The delay simulator can then use this clock to advance the contents of the buffer.

7.3 Recording Data

A simple Java applet was written to facilitate the recording of interaction data with inTouch-3. The applet assumes that two serial ports are available on the computer and implements two threads to receive incoming data. Each serial port taps data from the inTouch communication cable travelling in a single direction. Thus, one serial port captures data flowing from node 1 to node 2, while the other serial port captures data flowing in the other direction. The encoder data, which is binary, is made human-readable by conversion to hexadecimal ASCII when it is received from the serial port. These hexadecimal values are then written to a disk file for later analysis. A custom serial adapter was made to permit access to the data lines of port 1 in the inTouch communication cable. These two lines are connected to the receive data lines of the two serial ports of the computer.

8 Future Work

8.1 Custom Servo Amplifier

In retrospect, it would have been possible to design hardware to perform motor control equivalent in functionality to the Copley amplifier. This would have been a more cost effective solution in the long run.

One option is to use the PWM output of the PIC microcontroller to generate an analog output by passing the PWM waveform through a simple RC filter as shown in Figure 8.1. The time constant τ=RC should be chosen large enough to smooth out ripple in the analog output and is dependent on the PWM switching frequency. The voltage follower is added to ensure that the output has low impedance and does not load circuits connected to it.

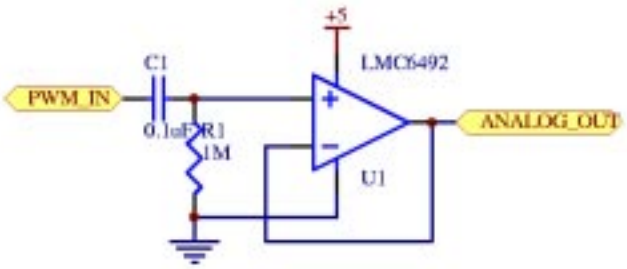


Figure 8.1: Generating an analog voltage from the PWM output

The power amplifier circuit to drive the motors might be similar to that shown in Figure 8.2. The simplest implementation uses bipolar voltage supplies as shown in the schematic. Power bipolar junction transistors are used as voltage-controlled current sources to source and sink current through the load as necessary. Feedback is used to correct for crossover distortion caused by the diode drops in the power transistors as well as for resistive loss through the load. The gain may be adjusted by the potentiometer. Interfacing the circuits of Figures 8.1 and 8.2 might prove to be a solution to replace the Copley amplifiers.

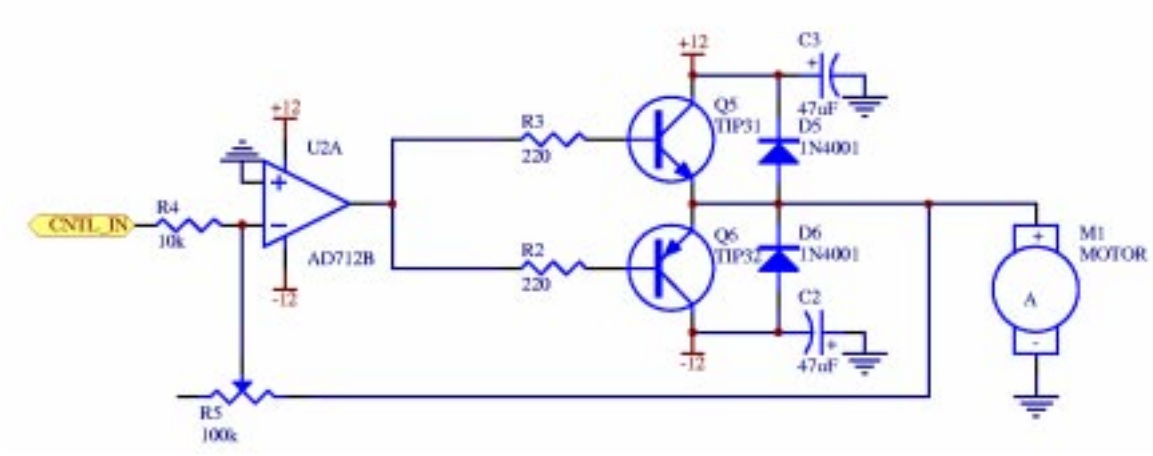


Figure 8.2: Power amplifier circuit

Another alternative may be to use the National Semiconductor LM12, a monolithic power operational amplifier designed to drive high current loads. The LM628 motion controller datasheets provide circuits for motor drive using the LM12. The reader is referred to these documents for more information. It should be noted that the optimal solution is to perform torque control in hardware with current feedback to off-load the microcontroller, which must execute the inTouch synchronization algorithm.

8.2 Power Supply Isolation

By its very nature, the servo amplifier generates a great deal of electrical noise through PWM switching of high currents. This electrical noise may be hazardous to logic on the controller board, or may cause improper operation at the very least. One method of reducing electrical noise propagation into the logic is through power supply isolation. In the current implementation of inTouch-3, both the servo amplifier and the controller board share the same 24 volt power supply. While the controller board is isolated from noise on the 24 volt line by means of the switching regulator of Figure 3.13, the logic ground is not. To suppress noise from the servo amplifier, a 1 μH inductor should be added between logic ground and the servo amplifier ground. The inductor need not be large, since it need only handle the small amount of current that the logic draws.

8.3 Controller Board Synchronization

In Chapter 6, the implementation of the modem interface unit was discussed. Recall that the test results showed that the modem bandwidth was insufficient for the inTouch-3 at the present sampling rate of 1kHz. The sampling rate might be reduced to 500Hz to cut the data rate in half. This change, however, would require readjustment of the algorithm parameters.

Alternatively, the controller boards could be synchronized in their sampling times, so that the data produced would already be in order and non-overlapping in time. Synchronization would also require a change in the sampling time, but it eliminates the need for the serial multiplexer/demultiplexer. A preliminary effort was made to synchronize the controller boards by wiring the three boards in a cascade configuration as shown in the figure below.

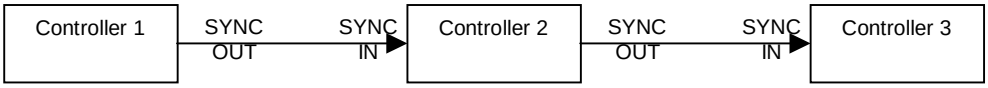


Figure 8.3: Controller board configuration for synchronization

The timing diagram of the configuration is shown here.

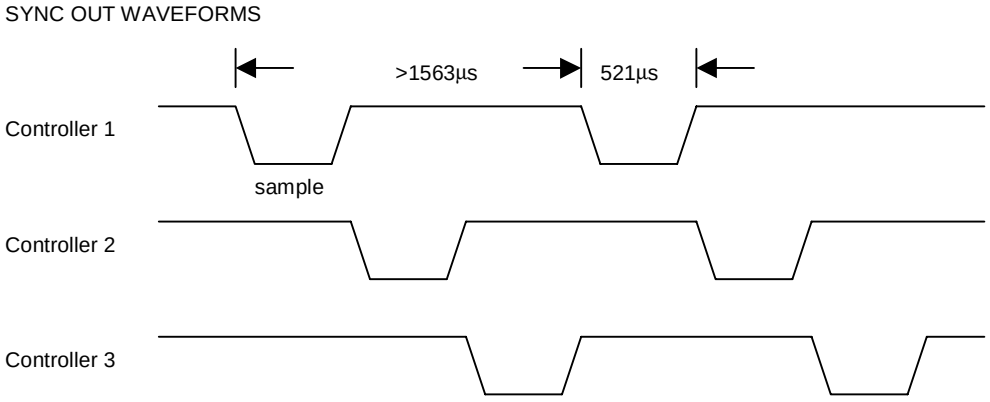


Figure 8.4: Timing diagram for inTouch-3 synchronization

The total processing time for each sample has been found empirically to be 380 µs. This figure can also be derived by counting the number of instruction cycles required to process each sample and multiplying by the instruction time (200 nanoseconds at 20

MHz). Another factor is the byte transmission time. Recall from Section 3.7 that at 19200 baud, this time is 521µs. Since the data transmission time exceeds the processing time, 521µs defines the lower limit of the sampling period to achieve non-overlapping data.

The sampling period, as shown in Figure 8.4, begins on the falling edge of the synchronization clock. Controller 1 serves as the master controller and generates the synchronization clock (SYNC OUT) as shown. After 521µs has elapsed, during which sampling, calculations, and serial communications has taken place, the rising edge of the synchronization clock triggers the sampling period of controller 2 through SYNC IN. Again after 521µs has elapsed, controller 2's SYNC OUT line triggers controller 3. Since controller 1 does not know when controller 3 completes, the period of the synchronization clock period is set greater than 1563µs to ensure that controller 1 does not begin transmitting until controller 3 has completed. As an alternative, controller 3 can be wired to controller 1 to provide this information.

This preliminary work has not been completed at the time of this writing, as there are a number of remaining difficulties to work through. The task of synchronizing the controller boards is reserved for future work.

8.4 Delay Compensation

Techniques for delay compensation and a quantitative study of the effects of delay on the system should be studied in future work. Using the system models presented in Section 4.4, the algorithm parameters can be varied to produce instability. The sample delay can likewise be adjusted to produce instability. Preliminary work has been done to compensate for delay using prediction and adaptive compensation for the inTouch-2 system [2]. Delay compensation for the inTouch-3 system is particularly important because the system is designed to be incorporated into other communication technologies as a self-contained unit. Delay of some form is present on any communications channel and so it is imperative that the problem be addressed.

8.5 Multi-Node Configuration

The inTouch-3 system at present supports two node interactions. To make the system more general, work should be done to develop multi-node support. There are various node topologies that may be explored. A ring topology as illustrated in the figure below is one possibility. The main advantage of the ring topology is that minimal changes to hardware and the synchronization algorithm are required. Since communication between nodes is point-to-point, the existing RS-232C interface may be used. The disadvantage of the ring topology is that communication delay is different over the two directions of data travel between nodes. Consider communication between node 1 and node 2. Data traveling from node 1 to node 2 encounters only a single unit delay. In contrast, data traveling from node 2 to node 1 encounters a three unit delay since it must first pass through node 3 and node 4 before arriving at node 1. While the synchronization algorithm need not be changed to support the ring topology, delay compensation will most likely need to be used to account for this difference in communication latency.

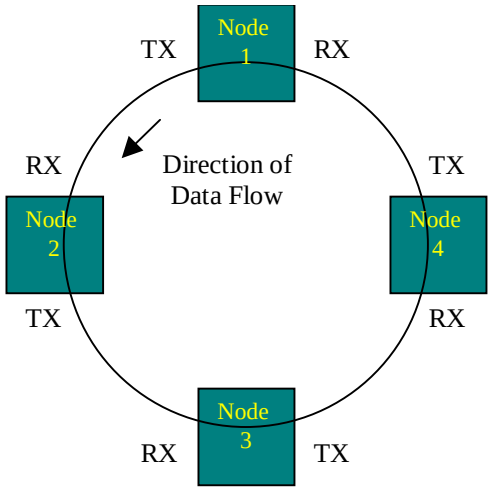


Figure 8.5: Four-node ring topology

A second possible topology is the shared bus of Figure 8.6. The main advantage of the shared bus topology is that communication latency between nodes is generally equal for short bus lengths. One disadvantage of this topology is that communication is not point-to-point as in the ring topology, but multi-master. In hardware, a multi-drop protocol such as RS-485 must be used. It is straightforward to adapt the RS-232 interface to support RS-485 with an interface device such as the Maxim MAX485. In software, the issues of collision detection and bus arbitration must be addressed, since

many nodes are transmitting on the same bus simultaneously. Moreover, the synchronization algorithm must be altered to account for multi-node input, perhaps by averaging the received positions.

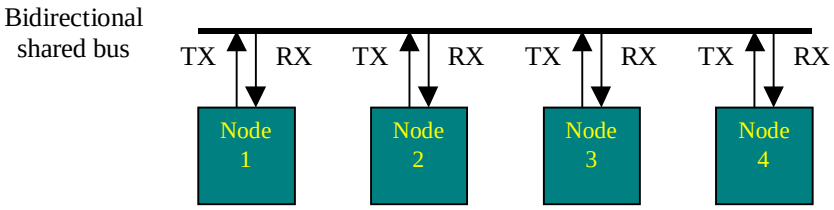


Figure 8.6: Four-node shared bus topology

9 Conclusions

This thesis describes the design and implementation of the inTouch-3, a self-contained, robust embedded control system for the purposes of communication by haptic force-feedback. Both hardware and firmware design have been discussed in detail, along with the design tradeoffs made in the process. The product of this development work is a system that is connected by a direct serial line and that is ready for distribution and use.

Several accessories have been developed including a module for interfacing the system to a modem. Initial tests showed that the modem bandwidth was insufficient to handle the data rate of the current inTouch implementation with data multiplexing. Two possible solutions to this problem include reducing the sampling rate from 1kHz and synchronizing the controller boards to eliminate the need for data multiplexing. Together, these techniques should allow the inTouch-3 to operate over a telephone line.

Another area for future development is in the synchronization of more than two objects. Using the ring topology described in the last section, only minimal changes to the firmware are required to adapt the inTouch-3 system to multi-node operation.

One foreseeable problem that may prevent inTouch-3 from operating as described is the problem of communication latency. This problem should be addressed in the future using the system models of Section 4.4. Once the effect of latency is understood, the system can be compensated appropriately to operate properly in any configuration.

Finally, this thesis has examined many issues regarding small-scale production. The results here may provide a framework for future production efforts of inTouch as well as other projects. As of January 1999, six sets of inTouch-3 systems are in the process of being distributed to various sponsors for field use. Another four sets will remain at the Media Laboratory for demonstration and experimental purposes.

It has been shown that the inTouch is a medium for physical communication. Given that there is no standard "language" of touch, it is interesting to observe how users create their own forms of communicate through this medium. It is hoped that the result of this distribution and experimentation will be further insight into the cognitive science aspect of the project, especially in the form of user studies. The method for data collection from the system as described above should be useful to this end.

10 References

1. Bertsekas, D. and Gallager, R. *Data Networks*. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1992.

2. Brave, S. *Tangible Interfaces for Remote Communication and Collaboration*. Master's Thesis in Media Arts and Sciences, Massachusetts Institute of Technology, 1998.

3. Brave, S. and Dahley, A. InTouch: A Medium for Haptic Interpersonal Communication. *Extended Abstracts of CHI '97* (Atlanta, March 1997), ACM Press, 363-364.

4. Fogg, B.J., Cutler, L., Arnold, P., and Eisback, C. HandJive: A Device for Interpersonal Haptic Entertainment, to appear in *Proceedings of CHI '98* (Los Angeles, April 1998), ACM Press.

5. Goldie, J. *Ten Ways to Bulletproof RS-485 Interfaces*. National Semiconductor application note 1057, 1996.

6. Gould, L.A., Markey, W.R., Roberge, J.K., Trumper, D.L. *Controls Systems Theory*. Not published. 2nd revision, 1997.

7. Hannaford, B. Kinesthetic Feedback Techniques in Teleoperated Systems. *Advances in Control and Dynamic Systems*. C. Leondes, ed. San Diego: Academic Press, 1991.

8. Horowitz, P. and Hill, W. *The Art of Electronics*. 2nd ed. New York: Cambridge University Press, 1990.

9. Hunt, Steven. *LM628 Programming Guide*. National Semiconductor application note 693, 1998.

10. Ishii, H., Kobayashi, M. and Arita, K. Iterative Design of Seamless Collaboration Media. *Communications of the ACM (CACM)* (1994). ACM Press, 37, 8, 83-97.

11. Linear Technologies LT1073 datasheet.

12. *LM628/629 User Guide*. National Semiconductor application note 706, 1993.

13. Martin, F. *The 6.270 Robot Builder's Guide*. Not published. 1992.

14. Maxim Integrated Circuits MAX1044 datasheet.

15. Microchip Technologies PIC16C7X datasheet.

16. National Semiconductor LM628 datasheet.

17. National Semiconductor LM18200 datasheet.

18. National Semiconductor LM2574 datasheet.

19. Oppenheim, A.V., Willsky, A.S., Young, I.T. *Signals and Systems*. Englewood Cliffs: Prentice-Hall, 1983.

20. Regan, Tim. *A DMOS 3A, 55V, H-Bridge: The LM18200*. National Semiconductor application note 694.

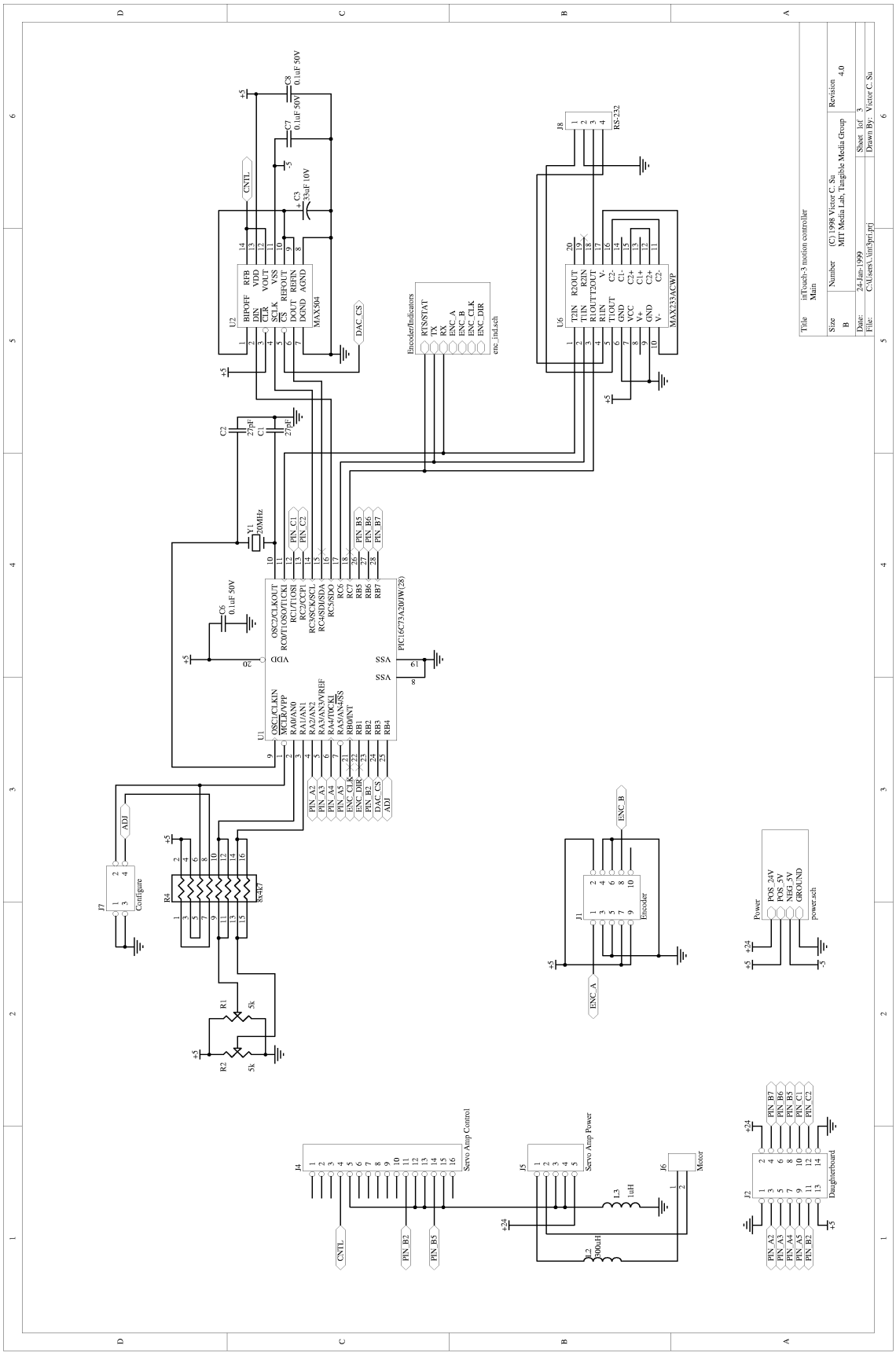
21. Schena, B. *Design of a Global Network of Interactive, Force-Feedback Sculpture*. Master's Thesis in the Department of Mechanical Engineering, Stanford University, 1995.

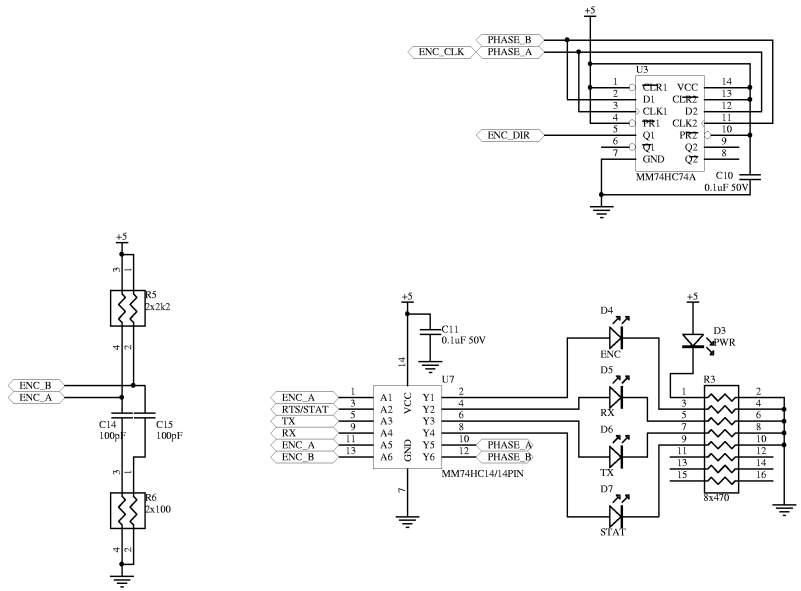
22. Strong, R., and Gaver, B. Feather, Scent and Shaker: Supporting Simple Intimacy. *Proceedings of Computer Supported Cooperative Work (CSCW) '96* (November 1996), 29-30.

23. *Understanding and Minimizing Ground Bounce*. National Semiconductor application note 640, 1993.

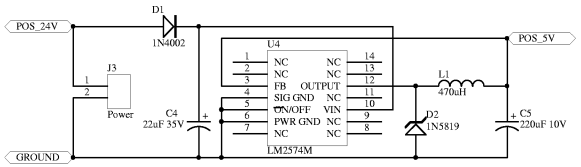
Appendix A: Circuit Schematics

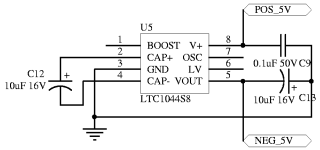
This section contains circuit schematics for the inTouch controller board, the mechanical unit auxiliary board, the modem interface unit, the port adapter, and the delay simulator.


Title: inTouch-3 motion controller
Main
Size: B
Number:
(C) 1998 Victor C. Su
MIT Media Lab, Tangible Media Group
Revision: 4.0
Date: 24-Jan-1999
Sheet 1of 3
File: C:\Users\.\in3prt.prj
Drawn By: Victor C. Su
U1
PIC16C73A20/JW(28)
U2
MAX504
U6
MAX233ACWP
X1
20MHz
C1
27pf
C2
27pf
C3
33uF 10V
C6
0.1uF 50V
C7
0.1uF 50V
C8
0.1uF 50V
R1
5k
R2
5k
R4
4x4k7
J1
Encoder
J2
Daughterboard
J4
Servo Amp Control
J5
Servo Amp Power
J6
Motor
J7
Configure
J8
RS-232
L2
500uH
L3
1uH
EncoderIndicators
enc_ind.sch
power.sch
Power
Encoder
Servo Amp Control
Servo Amp Power
Motor
Daughterboard
Configure
RS-232
CNTL
ADJ
DAC_CS
ENC_A
ENC_B
PIN_A2
PIN_A3
PIN_A4
PIN_A5
PIN_B2
PIN_B5
PIN_B6
PIN_B7
PIN_C1
PIN_C2
+5
+24
-5

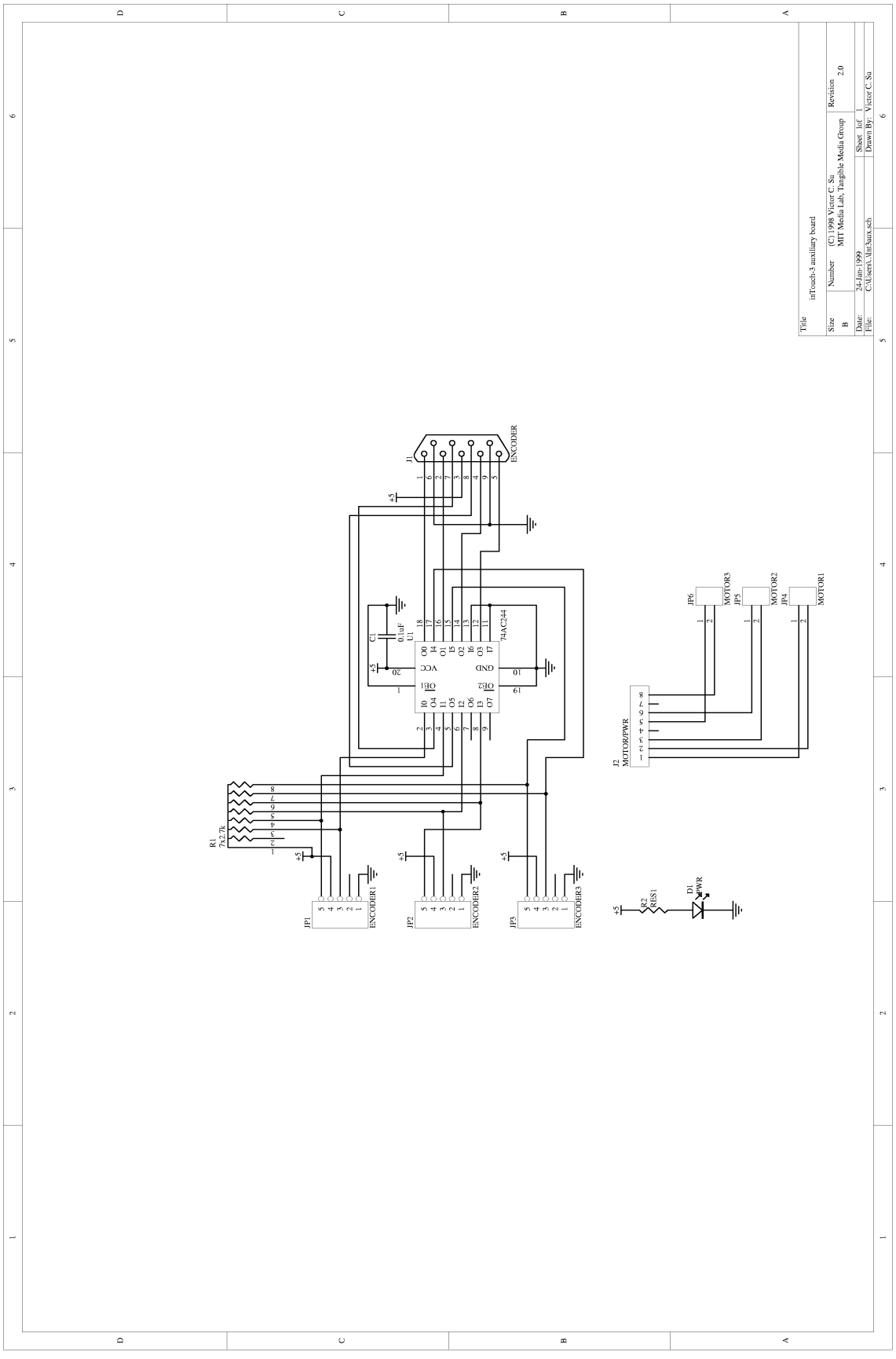


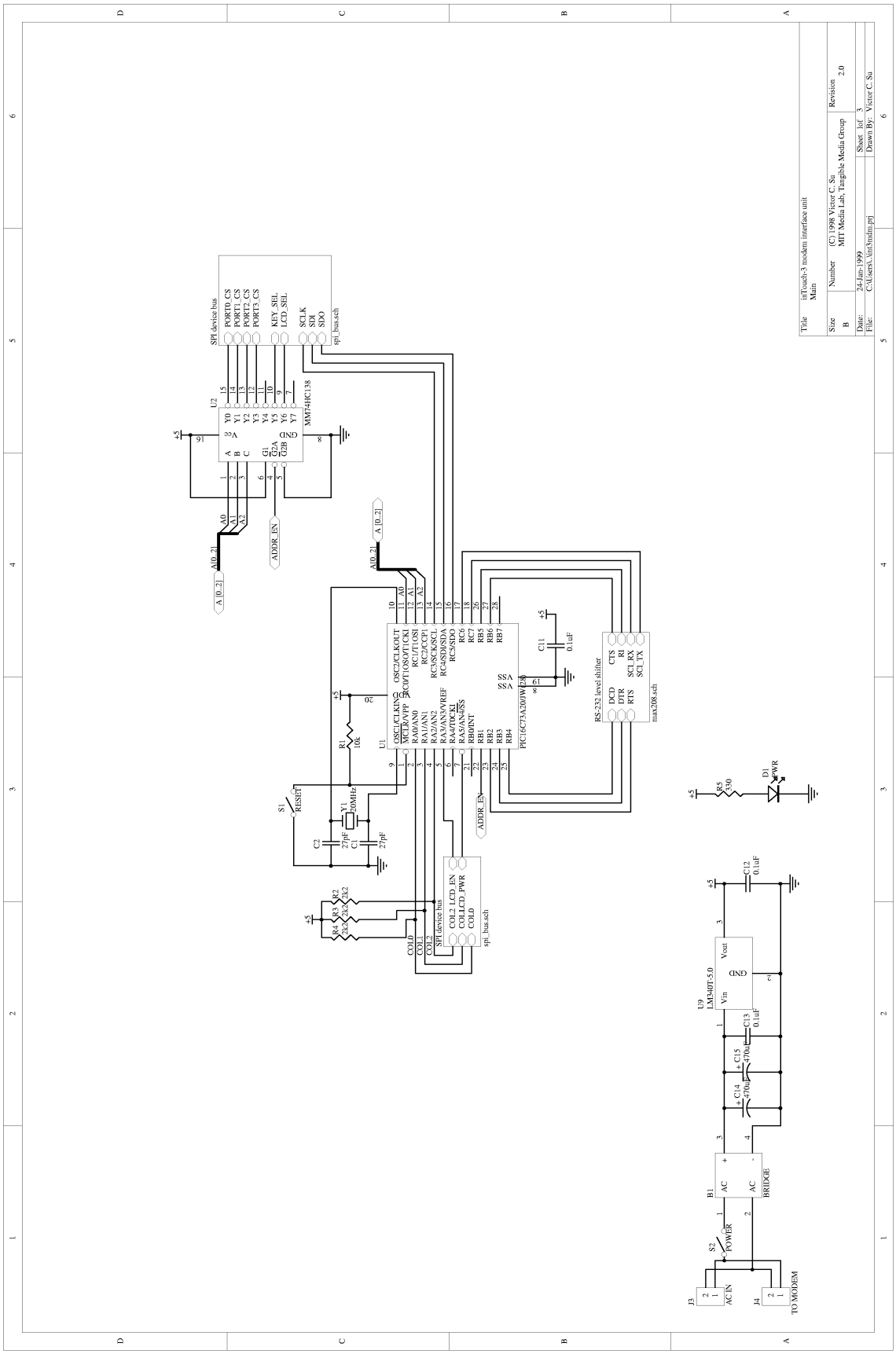
Title	inTouch-3 motion controller Encoders and indicators		
Size B	Number	(C) 1998 Victor C. Su MIT Media Lab, Tangible Media Group	Revision 4.0
Date:	24-Jan-1999	Sheet 2of 3	
File:	C:\Users\..\enc_ind.sch	Drawn By: Victor C. Su	

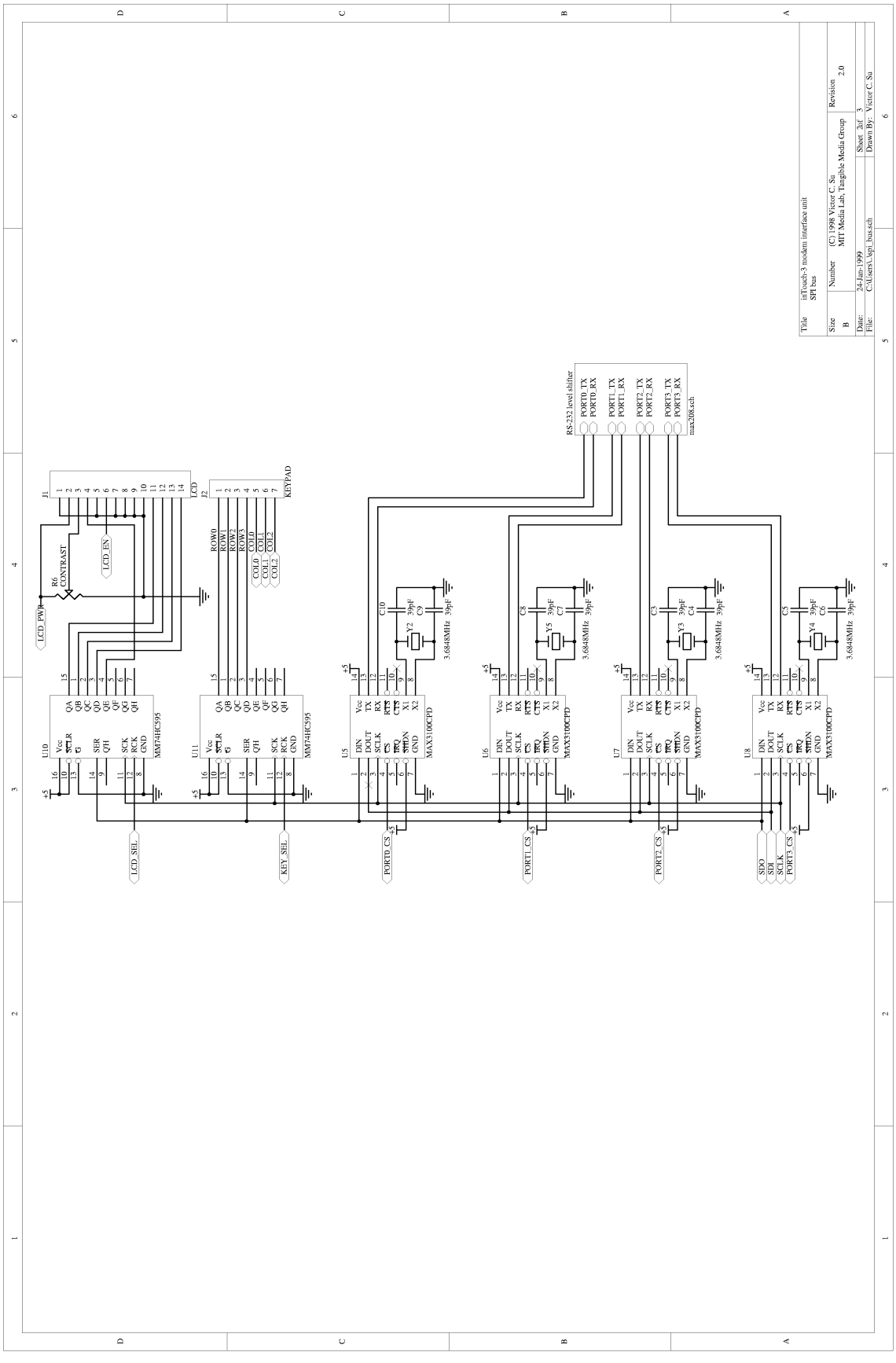


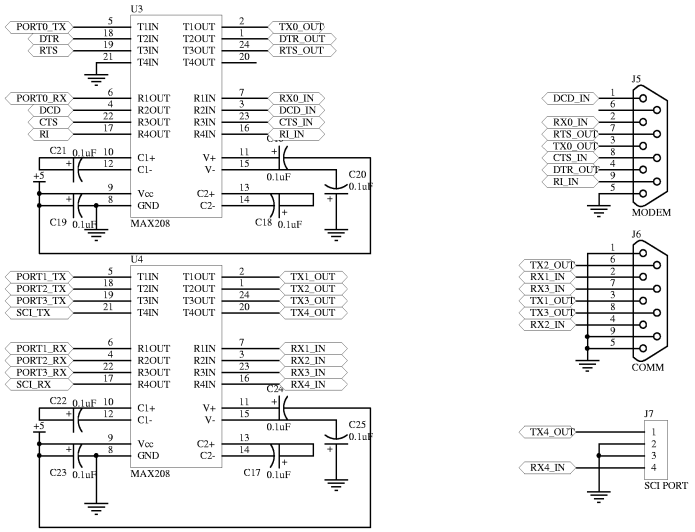


Title	inTouch-3 motion controller Power		
Size B	Number	(C) 1998 Victor C. Su MIT Media Lab, Tangible Media Group	Revision 4.0
Date:	24-Jan-1999		Sheet 3of 3
File:	C:\Users\..\power.sch		Drawn By: Victor C. Su

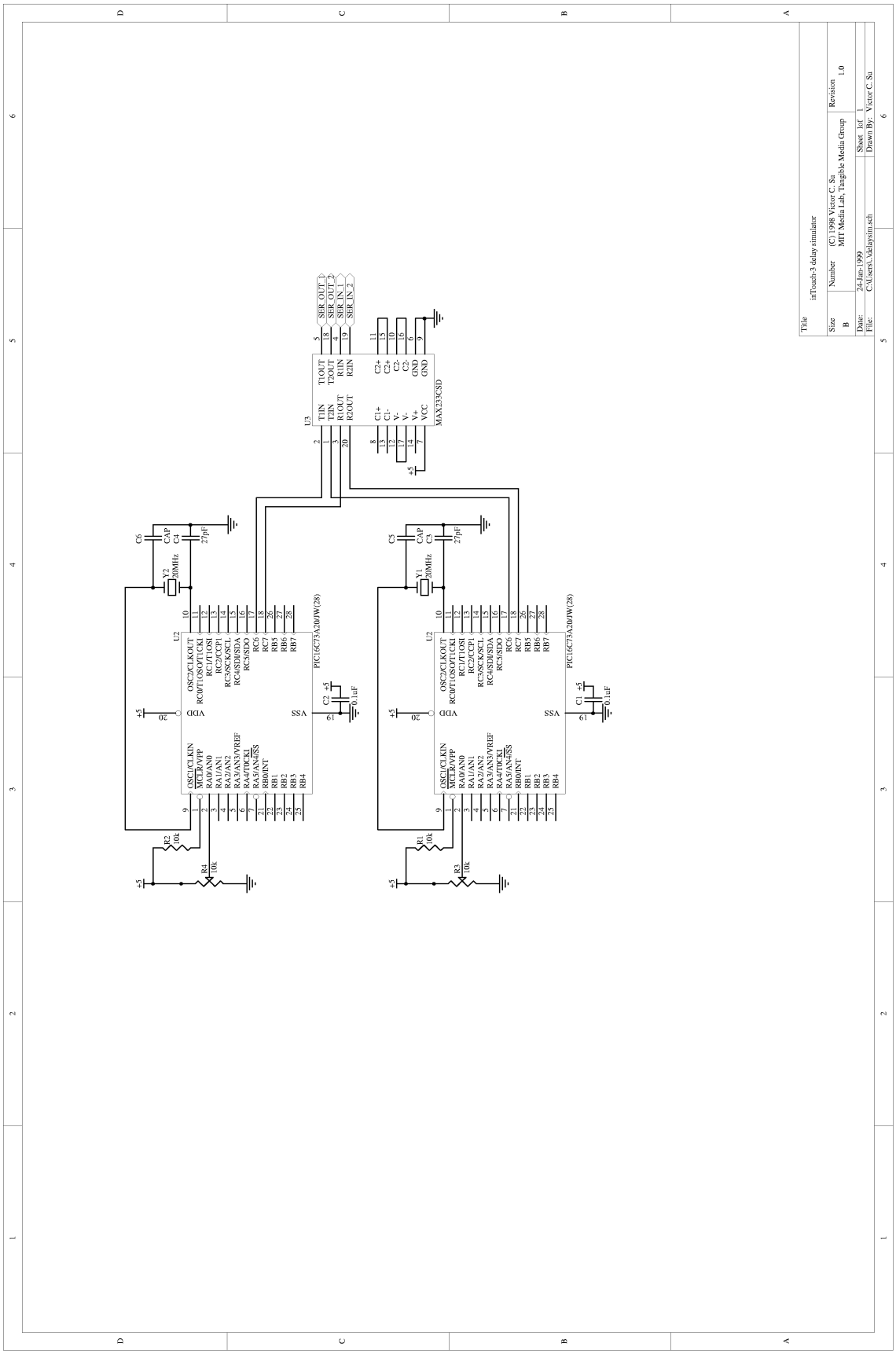

Title: inTouch-3 auxiliary board
Size: B
Number: (C) 1998 Victor C. Su
MIT Media Lab, Tangible Media Group
Revision: 2.0
Date: 24-Jan-1999
Sheet 1of 1
File: C:\Users\..\inTouch.sch
Drawn By: Victor C. Su
ENCODER
MOTOR1
MOTOR2
MOTOR3
MOTOR/PWR
74AC244
0.1uF
7k2.7k
RES1
PWR
ENCODER1
ENCODER2
ENCODER3
+5

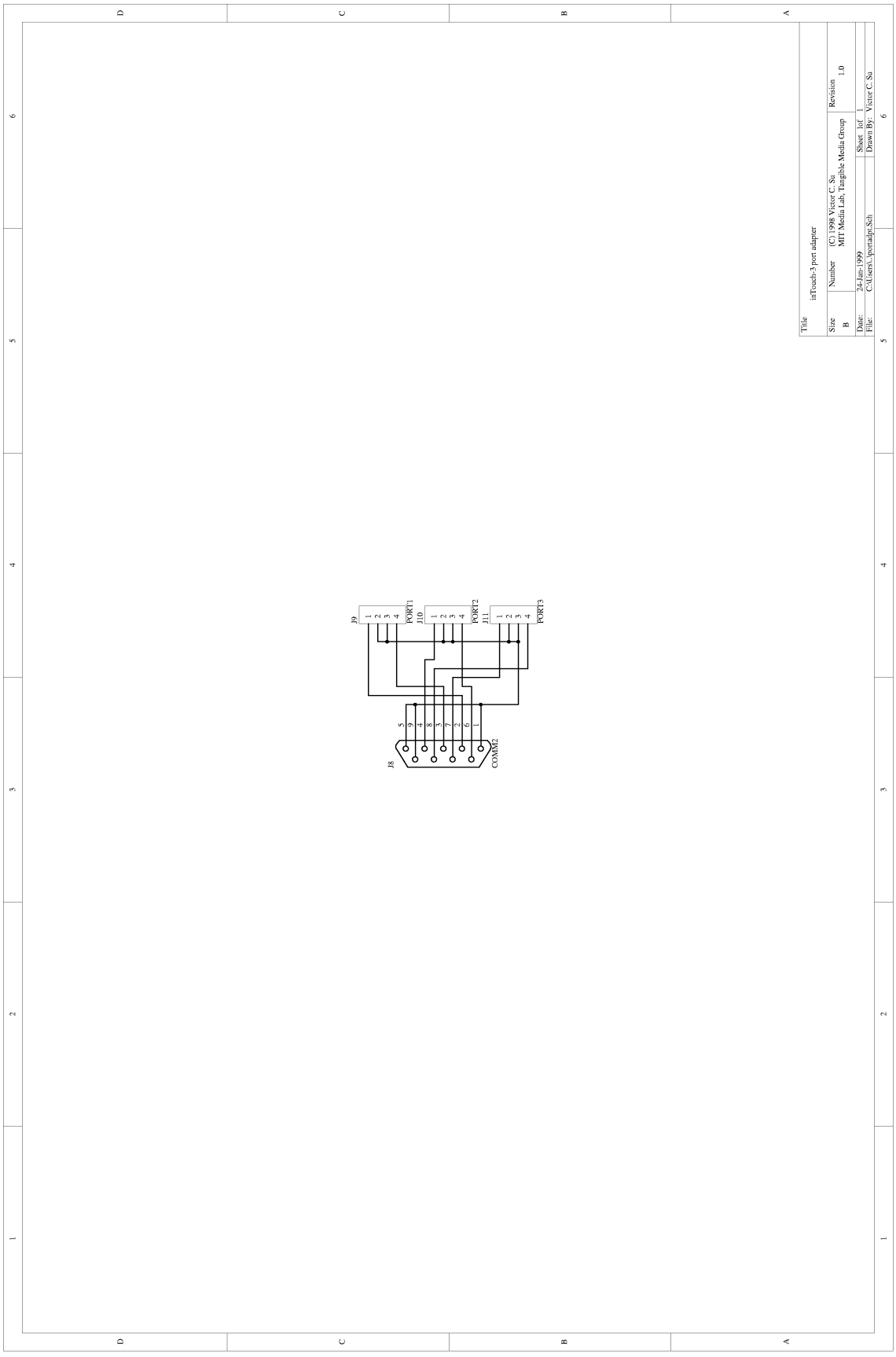

Title: inTouch-3 modem interface unit
Main
Size: B
Number:
(C) 1998 Victor C. Su
MIT Media Lab, Tangible Media Group
Revision: 2.0
Date: 24-Jan-1999
Sheet 1of 3
File: C:\Users\..\int3sndm.prj
Drawn By: Victor C. Su
SPI device bus
PORT0_CS
PORT1_CS
PORT2_CS
PORT3_CS
KEY_SEL
LCD_SEL
SCLK
SDI
SDO
spi_bus.sch
U2
MM74HC138
A0
A1
A2
ADDR_EN
A[0..2]
A[0..2]
U1
PIC16C73A20/IWF28
OSC1/CLKIN
OSC2/CLKOUT
MCLR/VPP
RA0/AN0
RA1/AN1
RA2/AN2
RA3/AN3/VREF
RA4/T0CKI
RA5/AN4/SS
RB0/INT
RB1
RB2
RB3
RB4
RC0
RC1
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RB5
RB6
RB7
VSS
VSS
R1
10k
Y1
20MHz
C2
27pF
C1
27pF
S1
RESET
R2
2k2
R3
2k2
R4
2k2
COL0
COL1
COL2
COL2_LCD_EN
COLLCD_PWR
COL0
C11
0.1uF
RS-232 level shifter
max208.sch
CTS
RI
SCL_RX
SCL_TX
DCD
DTR
RTS
R5
330
D1
PWR
U9
LM340T-5.0
Vin
Vout
GND
C12
0.1uF
C13
0.1uF
C15
470uF
C14
470uF
B1
BRIDGE
AC
AC
S2
POWER
J3
AC IN
J4
TO MODEM


Title: inTouch-3 modem interface unit
SPI bus
Size: B
Number:
(C) 1998 Victor C. Su
MIT Media Lab, Tangible Media Group
Revision: 2.0
Date: 24-Jan-1999
Sheet 2of 3
File: C:\Users\ \spi_bus.sch
Drawn By: Victor C. Su
RS-232 level shifter
max206.sch
PORT0_TX
PORT0_RX
PORT1_TX
PORT1_RX
PORT2_TX
PORT2_RX
PORT3_TX
PORT3_RX
J1
J2
LCD
KEYPAD
R6
CONTRAST
LCD_PWR
LCD_EN
ROW0
ROW1
ROW2
ROW3
COL0
COL1
COL2
U10
U11
MM74HC595
U5
U6
U7
U8
MAX3100CPD
Y2
Y3
Y4
Y5
3.6848MHz
39pF
LCD_SEL
KEY_SEL
PORT0_CS
PORT1_CS
PORT2_CS
PORT3_CS
SDO
SDI
SCLK



Title	inTouch-3 modem interface unit RS-232 converter		
Size B	Number	(C) 1998 Victor C. Su MIT Media Lab, Tangible Media Group	Revision 2.0
Date:	24-Jan-1999		Sheet 3of 3
File:	C:\Users\..\max208.sch		Drawn By: Victor C. Su


Title
irTouch-3 delay simulator
Size
B
Number
Revision
1.0
(C) 1998 Victor C. Su
MIT Media Lab, Tangible Media Group
Date:
24-Jan-1999
Sheet 1of 1
File:
C:\Users\..\delaysim.sch
Drawn By: Victor C. Su
MAX233/SD
PIC16C775A20/W(28)
20MHz
CAP
27pF
0.1uF
10k
SHR_OUT_1
SHR_OUT_2
SHR_IN_1
SHR_IN_2


Title
inTouch-3 port adapter
Size
B
Number
Revision
1.0
(C) 1998 Victor C. Su
MIT Media Lab, Tangible Media Group
Date:
24-Jan-1999
Sheet 1of 1
File:
C:\Users\..\portadpt.Sch
Drawn By:
Victor C. Su
J9
PORT1
J10
PORT2
J11
PORT3
J8
COMM2

Appendix B: Microcontroller Code

inTouch Controller

The following listing is the microcontroller code for the inTouch controller. The code is written in C for the Custom Computer Services (CCS) PIC C compiler. Note that the code is organized into libraries to facilitate reuse between different projects. Libraries are added using the compiler's *#include* directive as shown below. Since the libraries contain code to handle many different configurations, a complete code listing of the inTouch controller is not possible in a small space. Thus, only the top-level file is included here, and the reader is referred to the accompanying CD-ROM for the complete listing.

```
/*
 *  TOUCH25B.C
 *  inTouch-3 firmware
 *
 *  Motor position controller with RS-232 serial communications interface
 *
 *  Device: PIC16C73A/PIC16C76
 *  Resource allocation:
 *    port A (RA0) : Gain 1 adjustment
 *           (RA1) : Gain 2 adjustment
 *           (RA2) : Reverse direction mode select out
 *           (RA3) : Synchronous sampling mode select out
 *           (RA4) : Synchronous clock input
 *           (RA5) : Synchronous clock output
 *    port B (RB0) : Encoder clock input
 *           (RB1) : Encoder direction input
 *           (RB2) : Servo amplifier drive enable
 *           (RB3) : DAC chip select (CS)
 *           (RB4) : Adjustment mode select
 *           (RB5) : Servo amplifier fault detect
 *           (RB6) : Synchronous sampling mode select in
 *           (RB7) : Reverse direction mode select in
 *    port C (RC0) : RS-232 RTS/status LED
 *           (RC3) : SPI clock
 *           (RC4) : SPI data in
 *           (RC5) : SPI data out
 *           (RC6) : RS-232 transmit data
 *           (RC7) : RS-232 receive data
 *
 *  Note that serial RTS shares the same pin as the status indicator LED.
 *
 *  Installing the jumper on the controller board at initialization
 *  results in adjustment mode.  The ADC gains are printed to serial
 *  output and resampled every half second.  The DAC output is also
 *  zeroed so the servo amplifier's offset can be trimmed.  Removing
 *  the jumper resumes initialization.  If the jumper is installed while
 *  in run (resample) mode, then derivative feedback gain and anti-damping
 *  gain may be adjusted dynamically.  Otherwise, both gains are sampled
 *  once and set at initialization.  Derivative gain is mapped to gain
 *  adjustment 2.  Anti-damping gain is mapped to gain adjustment 1.
 *
 *  The stall force limit is set so that the motor draws its maximum rated
 *  current at stall.  The servo amplifier is configured at a transconductance
 *  of 0.7A/V and the output of the DAC is approximately -2V to 2V.  The range
 *  of a 10-bit DAC is [-511, 511].  Since the Maxon motors are rated at 1.25A
 *  continuous, the force limit is set to a magnitude of 450 (i.e. -450 to 450)
```

```
 *  to produce -1.8V to 1.8V DAC output, and thus -1.25A to 1.25A drive current.
 *
 *  If the system fails to initialize, the processor will auto-reset in
 *  2.304 seconds by the watchdog timer (WDT).  If the processor stalls for
 *  any reason during resampling mode, it will auto-reset in 18ms, again by
 *  the watchdog timer.  Restart_wdt() must be called before this interval
 *  during normal operation to prevent unintentional reset.
 *
 *  Status LED Flash Codes:
 *    LED      Mode, State          Meaning
 *   ---------------------------------------------------------------------
 *    ENC      Run, Flash           Encoder activity
 *    TX       Run, Flash           RS-232 transmit data
 *    RX       Run, Flash           RS-232 receive data
 *    STAT     Init, Off            Adjustment mode
 *    STAT     Init, On             Sampling (run) mode
 *    STAT     Init, Blinking       Servo controller initialization failure
 *
 *  See INTOUCH.LOG for detailed revision history.
 *
 *  Modification Log
 *  ========================================================
 *    2.4c   1 Jun 1998  vcs
 *    2.4d  15 Jun 1998  vcs  added auto-reset
 *    2.4e  22 Jun 1998  vcs  changed WDT routines
 *    2.4f  23 Jun 1998  vcs  reorganized modules
 *    2.4g   8 Jul 1998  vcs  added reverse direction mode
 *                            added anti-friction filter
 *    2.4h  25 Aug 1998  vcs  reorganized resampling routine
 *    2.4i   1 Sep 1998  vcs  added fault detect routines
 *    2.4j  17 Sep 1998  vcs  optimization changes
 *    2.4k  17 Sep 1998  vcs  moved resampling into main loop
 *    2.5a  17 Sep 1998  vcs  moved resampling back to INT_RTCC
 *                            added synchronous sampling mode
 *    2.5b  26 Dec 1998  vcs  removed PSC code, new PID_Filter params
 *
 */


#fuses HS, WDT, NOPROTECT

#include <16C73A.H>                     // Uncomment to use the 16C73A device
//#include <16C76.H>                    // Uncomment to use the 16C76 device

#use delay(clock=20000000, RESTART_WDT)  // 20 MHz clock frequency
#use fast_io(a)
#use fast_io(b)
#use fast_io(c)

#priority RTCC, RDA, EXT                // Set interrupt handling priority

// Set software switches
#define DAC_MAX504
#define SINGLE_HW_ENC

#include "touch.h"                      // Main header file
#include "quad_enc.c"                   // Quadrature encoder library
#include "filter.c"                     // Digital filter library
#include "asc.c"                        // Analog servo controller functions

#define VERSION "2.5b"                  // Version information



// ============================================================================
// Initialization procedure
//

void Initialize() {
   int count;
   int check;
```

```
// Turn off interrupts
disable_interrupts(GLOBAL);

// Set tri-state for ports
set_tris_a(0x13);
set_tris_b(0xF3);
set_tris_c(0x90);

// Turn off status LED
DEASSERT_STAT_LED;

// Turn on pull-ups on port B
port_b_pullups(TRUE);

// Configure timer 0 initially for WDT
// Must call restart_wdt() at least every 2304ms to prevent reset
setup_counters(RTCC_INTERNAL, WDT_2304MS);

// Initialize the analog servo controller
check = Init_ASC();

if (check == 0) {  // 0 is successful return code

   // Initialize ADC for digital filters
   Init_ADC_Gain();

   // Set encoder direction
   output_low(DIR_SEL_OUT);
   if (!input(DIR_SEL_IN)) {   // Jumper set, reverse direction mode
      Set_Enc_Rev_Dir(TRUE);
      Set_ASC_Rev_Dir(TRUE);
   }

   // Set sampling mode
   output_low(SYNC_SEL_OUT);
   if (!input(SYNC_SEL_IN)) {  // Jumper set, synchronous sampling mode
      sync_mode = 1;
   }

   // Version information and parameter adjustment
   if (ADJ_MODE) {
      Print_Version();
      while (ADJ_MODE) Gain_Adjust();
      Init_ADC_Gain();    // Sample gains again and set
   }

   // Reconfigure RTCC according to sampling mode
   if (sync_mode) {
      // Synchronous sampling mode, external clock
      setup_counters(RTCC_EXT_L_TO_H, WDT_2304MS);
      set_rtcc(RTCC_START_SYNC);
   } else {
      // Asynchronous sampling mode, internal clock
      // Set TIMER0 (RTCC) period (at 20MHz): 1/((20000000/4)/64) = 12.8 us
      // Must call restart_wdt() at least every 18ms to prevent reset
      setup_counters(RTCC_INTERNAL, RTCC_DIV_64);
      set_rtcc(RTCC_START_ASYNC);
   }

   // Indicate successful initialization
   ASSERT_STAT_LED;

} else {

   // Blink LED
   for (count = 0; count < 3; count++) {
      ASSERT_STAT_LED;
      delay_ms(200);
      DEASSERT_STAT_LED;
      delay_ms(100);
```

```
      }

      // Output return code to serial
      printf("Init failure code: %u\n\r", check);

// Auto-reset on initialization failure via watchdog timer
      while(1);

   }

   // Initialize the encoder
   Initialize_Encoder();

   // Turn on interrupts
   enable_interrupts(INT_RDA);      // Enable serial data received interrupt
   enable_interrupts(INT_RTCC);
   enable_interrupts(GLOBAL);       // Activate interrupts
}


// Print version information
void Print_Version() {
   printf("INTOUCH-3 FIRMWARE\n\r");
   printf("Revision ");
   printf(VERSION);
   printf("\n\r");
   printf("(C) 1998 MIT Media Lab\n\r");
   printf("Tangible Media Group\n\r");
   printf("Sync mode: %u\n\r", sync_mode);
   printf("Enc prescaler: %u\n\r", ENC_PRESCALE);
   printf("Enc rev dir: %u\n\r", encoder_rev_dir);
   printf("ASC rev dir: %u\n\r", asc_rev_dir);
}



// ============================================================================
// Interrupt service routines (ISR)
//

#INT_RDA
// Service serial data when received
void Service_RDA() {
   remote_count = getc();  // Update remote position
}


#INT_RTCC
// This ISR is run every time the 8-bit RTCC rolls over and determines the
// sampling interval.
//
// In asynchronous sampling mode, execution occurs with period
// (255-RTCC_START_ASYNC)*(12.8us) and for a setting of RTCC_START_ASYNC = 177,
// the cycle time is 1.0 ms.
//
// In synchronous sampling mode, execution occurs whenever the external
// clock transitions from low to high.
//
void Service_RTCC() {
   if (!sync_mode) {
      set_rtcc(RTCC_START_ASYNC);
   } else {
      set_rtcc(RTCC_START_SYNC);
   }

   output_low(SYNC_CLK_OUT);

   if (DRV_READY) {     // Check for servo amp fault conditions
      restart_wdt();    // Reset watchdog timer
   } else {
      output_high(SYNC_CLK_OUT);
```

```
      while(1);         // Stop and reset via WDT
   }

   Resample();

   while(!TMRT);               // Wait until serial transmitter is finished
   output_high(SYNC_CLK_OUT);  // Trigger next controller
}



// ============================================================================
// Sampling procedure
//

// Resample counters, transmit, and call PID position controller
void Resample() {
   signed long force;
   signed long pos_error;

   // Sample local encoder counter
   local_count = encoder_counter;

   // Send to remote device if it has changed
   if (local_count != last_local_count) putc(local_count);

   // Find and accumulate local and remote position changes
   abs_local_pos  += Get_Counter_Change(local_count, last_local_count);
   abs_remote_pos += Get_Counter_Change(remote_count, last_remote_count);

   // Save for next iteration
   last_local_count  = local_count;
   last_remote_count = remote_count;

   // Dynamically adjust gains if external jumper set
   if (ADJ_MODE) Poll_Gains();

   // Apply a PID filter to the local and remote positions
   // Note that because of the way the position error is defined, if the
   // local position increases, the force will increase and be in the
   // opposing (negative) direction.
   pos_error = abs_remote_pos - abs_local_pos;
   force = PID_Filter(pos_error);

   // Apply anti-damping filter to reduce roller friction
   force += Anti_Damp_Filter(abs_local_pos);

   // Apply amplitude filter to limit the force and avoid overdriving the motor
   force = Amp_Filter(force);

   // Call the servo controller with calculated force
   // Verify the writes and reset via WDT on fault
   if (!Set_ASC(force)) while(1);
}



// ============================================================================
// main
//

void main() {
   Initialize();
   while(1);
}
```

Modem Interface Unit

The following is the top-level file for the modem interface unit. Again, the reader is referred to the accompanying CD-ROM for a complete listing.

```
/*
 *  MODEM22A.C
 *  inTouch-3 firmware
 *
 *  Modem communications adapter
 *  RS-232 serial 3-to-1 multiplexer/demultiplexer and modem controller
 *
 *  Device: PIC 16C73A
 *  Resource allocation map:
 *     port A (RA0) : Keypad column 1 input
 *            (RA1) : Keypad column 2 input
 *            (RA2) : Keypad column 3 input
 *            (RA3) : LCD enable
 *            (RA5) : LCD power
 *     port B (RB1) : Address decoder enable
 *            (RB2) : Request to send (RTS)
 *            (RB3) : Data terminal ready (DTR)
 *            (RB4) : Data carrier detect (DCD)
 *            (RB5) : Ring indicator (RI)
 *            (RB6) : Clear to send (CTS)
 *     port C (RC0) : Address decoder bit 0
 *            (RC1) : Address decoder bit 1
 *            (RC2) : Address decoder bit 2
 *            (RC3) : SPI clock
 *            (RC4) : SPI data in
 *            (RC5) : SPI data out
 *
 *  See MODEM.LOG for detailed revision history.
 *
 *  Modification Log
 *  ================================================================
 *    2.1b  18 May 1998  vcs
 *    2.1c  18 Jun 1998  vcs  modified UART functions
 *    2.1d  12 Jul 1998  vcs  reorganized modules
 *    2.2a  15 Sep 1998  vcs  optimized code
 *
 */


#fuses HS, NOWDT, NOPROTECT

#include <16C73A.H>
//#include <16C76.H>

#use delay(Clock=20000000)       // 20 MHz clock frequency
#use fast_io(a)
#use fast_io(b)
#use fast_io(c)

// Set software switches
#define MUX_EXT_PORT_IO
#define LCD_SW_RESET
#define KEY_VERIFY

#include "modem.h"               // Main header file
#include "lcdspi.c"              // LCD driver library
#include "keyspi.c"
#include "mux.c"                 // Data mux/demux module
#include "mdmcntl.c"             // Modem controller module

#define VERSION "2.2a"
```

```
// ============================================================================
// Top-level initialization procedure
//

void Initialize(void) {
   int check;

   // Power-up delay to protect against transients
   delay_ms(1000);

   // Turn off interrupts for initialization
   disable_interrupts(GLOBAL);

   // Set tri-state for ports
   set_tris_a(0x07);
   set_tris_b(0x70);
   set_tris_c(0x90);

   // Configure port A for digital I/O
   setup_port_a(NO_ANALOGS);

   // Turn on port B weak pull-ups
   port_b_pullups(TRUE);

   // Initialize SPI bus
   SPI_Bus_Deselect();

   // Initialize LCD
   check = Initialize_LCD();
   printf(Send_LCD, "\fLCD: ");
   if (check) {
      printf(Send_LCD, "Ok ");
   } else {
      printf(Send_LCD, "No ");
      printf("LCD failure\n\r");
   }

   // Initialize keypad
   check = Initialize_Keypad();
   printf(Send_LCD, "Key: ");
   if (check) {
      printf(Send_LCD, "Ok");
   } else {
      printf(Send_LCD, "No");
      printf("Keypad failure\n\r");
   }

   // Initialize mux
   check = Initialize_Mux();
   printf(Send_LCD, "\nMux: ");
   if (check == 4) {
      printf(Send_LCD, "Ok ");
   } else {
      printf(Send_LCD, "No ");
      printf("Port %u failure\n\r", check);
   }

   // Initialize modem
   check = Initialize_Modem();
   printf(Send_LCD, "Mdm: ");
   if (check) {
      printf(Send_LCD, "Ok");
   } else {
      printf(Send_LCD, "No");
      printf("Modem failure\n\r");
   }

   delay_ms(2000);
```

```
   // Print version information
   printf(Send_LCD, "\fINTOUCH-3  v");
   printf(Send_LCD, VERSION);
   printf(Send_LCD, "\nREADY");

   // Bring up interrupts
   enable_interrupts(GLOBAL);
}


// ============================================================================
// main
//

void main() {
   Initialize();
   while(1) {
      if (Poll_Keypress()) Process_Key_Input(Get_Key());
      if (event) Service_Link();
      if (linkState == CONNECTED) Route_Data();
   }
}
```

PWM Servo Controller

The following is the PWM servo controller code used in the initial tests. The PWM servo code was ultimately not used and was replaced by the Copley servo amplifier.

```
/*
 *  PSC_CLSW.C
 *  TMG PIC firmware library
 *
 *  PWM servo controller
 *  Closed loop control, software implementation
 *
 *  PIC information: device-independent
 *  Resource allocation:
 *    port A (RA3) : ADC feedback sense
 *    port B (RB2) : PWM direction output
 *           (RB3) : H-bridge drive mode select output
 *    port C (RC2) : PWM magnitude output
 *  Tri-state register configuration:
 *    RA3 : input
 *    RB2 : output
 *    RB3 : output
 *    RC2 : output
 *
 *  This file implements a PIC-based PWM servo controller.
 *  The LMD18200T H-bridge is assumed.
 *
 *  The ADC module must be configured properly for H-bridge current
 *  sense to work.  The H-bridge current sense resistor should be set
 *  so that 5V is generated at the maximum desired motor current.  The
 *  bridge produces 377uA per A of applied current.  The resistance is
 *  thus calculated as:  R = 5/(377uA * current limit).
 *
 *  Velocity control is also supported if a tachometer coupled to
 *  motor is used.  The tachometer should be connected to a full-wave
 *  rectifier so that only positive voltages are produced.
 *
 *  Two drive modes are supported: locked on zero force, and
 *  free on zero force.  In locked mode, the motor terminals are
```

```
 *  shorted during the off period of the duty cycle.  This means that
 *  the motor "holds" during operation, which is good for positioning
 *  applications.  Under free mode, the motor terminals are left
 *  open during the off period, thus allowing the motor to "slip" during
 *  operation.  This mode avoids the inductive friction characteristic
 *  of locked mode operation, and is better suited for tactile
 *  applications such as the inTouch.
 *
 *  Modification Log
 *  ============================================================
 *    1.0  19 Mar 1998  vcs  initial release
 *    1.1   1 Sep 1998  vcs  modified return code of init_psc()
 *    1.2  26 Dec 1998  vcs  modified to use digital filter library
 *
 */


#ifndef __PSC_CLSW__
#define __PSC_CLSW__

#include "filter.c"                // Digital filter library

#define PWM_DIR            PIN_B2   // PWM direction output
#define DRV_SEL            PIN_B3   // H-bridge drive mode select output

#define CURR_ADC_CHAN      3        // ADC channel for current sense
#define ADC_ACQ_DELAY_US   20       // ADC acquisition delay in microseconds

#define LOCKED_ON_ZERO     (output_high(DRV_SEL))
#define FREE_ON_ZERO       (output_low(DRV_SEL))

#define POS_PWM_DIR        (output_high(PWM_DIR))
#define NEG_PWM_DIR        (output_low(PWM_DIR))

#define POS_DRV_ASSERTED   (input(PWM_DIR))



// ============================================================================
// Forward declarations
//

void Set_PSC(signed long desired);
int Init_PSC(void);
int Poll_ADC(void);
signed long Get_Error(signed long desired_value);



// ============================================================================
// PWM servo controller functions
//

// This PWM servo controller performs torque or velocity feedback control
// using a digital PID filter to achieve zero error convergence.  The PWM
// duty cycle is dynamically adjusted to make the difference between the
// desired value and the actual value converge to zero.  The desired
// value has 8-bit resolution in two directions.
//
// Torque or velocity may be controlled depending what is connected
// to the ADC input.  For torque mode, connect the output of the current
// sense resistor to the ADC.  For velocity mode, connect the rectified
// output of the tachometer to the ADC.  Gains may need to be adjusted
// in the PID filter module.

void Set_PSC(signed long desired) {
   static signed long duty_cycle = 0;
   signed long error;

   // Resample error
   error = Get_Error(desired);
```

```
   // Apply PID filter to the error
   duty_cycle += PID_Filter(error);

   // Range check since duty cycle can only be 10 bits in magnitude
   if (duty_cycle > 1023)  duty_cycle = 1023;
   if (duty_cycle < -1023) duty_cycle = -1023;

   // Extract sign information to set the H-bridge direction input
   if (duty_cycle < 0) {
      NEG_PWM_DIR;             // Set negative PWM direction
   } else {
      POS_PWM_DIR;             // Set positive PWM direction
   }

   // Set the PWM magnitude
   set_pwm1_duty((unsigned long) duty_cycle);
}


// PSC initialization procedure
int Init_PSC() {

   // Hold the motor in place during initialization
   LOCKED_ON_ZERO;

   // Select PWM
   setup_ccp1(CCP_PWM);   // Configure CCP1 as a PWM
   setup_ccp2(CCP_OFF);   // Disable CCP2

   // Set PWM switching frequency
   // The cycle time will be (1/clock)*4*t2div*(pr2+1)
   // At 20MHz, clock=20000000, pr2=0xFF and t2div=1 (below)
   // so the cycle time is:
   // (1/20000000)*4*256*1 = 51.2 us or 19.53 kHz
   // CCP1 and CCP2 share the same frequency
   setup_timer_2(T2_DIV_BY_1, 0xFF, 0);

   // Set PWM duty cycle (10-bit resolution)
   set_pwm1_duty(0);

   // Release the motor drive
   FREE_ON_ZERO;

   return 0;
}


// ADC sampling procedure
// Delay is necessary for acquisition time after changing ADC channels
// Return ADC value
int Poll_ADC() {
   set_adc_channel(CURR_ADC_CHAN);
   delay_us(ADC_ACQ_DELAY_US);
   return(read_adc());
}


// The feedback value to the ADC may be the current sense
// output or it may be the rectified output of the tachometer
// for velocity control.
signed long Get_Error(signed long desired_value) {
   signed long error;
   signed long actual_value;
   int magnitude;

   // Range check desired value to limit to 8-bits in 2 directions
   if (desired_value > 255)  desired_value = 255;
   if (desired_value < -255) desired_value = -255;

   // Poll magnitude of actual value
```

```
   magnitude = Poll_ADC();

   // Examine drive direction so we can place a sign on the magnitude
   if (POS_DRV_ASSERTED) {
      actual_value = (signed long) magnitude;
   } else {
      actual_value = (signed long) -magnitude;
   }

   // Calculate error (current difference)
   error = desired_value - actual_value;

   return error;
}

#endif   // #ifndef __PSC_CLSW__
```

Delay Simulator

The following is a code listing for the delay simulator described in Section 7.1.

```
/*
 *  DELAY09A.C
 *  inTouch-3 firmware
 *
 *  RS-232 serial communications delay simulator
 *
 *  Device: PIC16C73A/PIC16C76
 *  Resource allocation:
 *    port C (RC6) : RS-232 transmit (TX)
 *           (RC7) : RS-232 receive  (RX)
 *
 *  Modification Log
 *  ================================================
 *    rev 0.9a     12 May 1998     vcs   pre-release
 *
 */

#fuses HS, NOWDT, NOPROTECT

#include <16C73A.H>                // Uncomment to use the 16C73A device
//#include <16C76.H>               // Uncomment to use the 16C76 device

// Set software switches
#define MULTIBANK_QUEUE

#include "queue.c"                 // Data queue library

#use delay(Clock=20000000)         // 20 MHz clock frequency
#use fast_io(a)
#use fast_io(c)

#define SER_TX         PIN_C6
#define SER_RX         PIN_C7
#define TRIS_A         0xFF
#define TRIS_C         0x81
#define ADJ_SEL        (input(PIN_C0))
#define VERSION        "0.9a"

// 3-line (TX/RX/GND) RS-232 protocol
#use rs232(baud=19200, xmit=SER_TX, rcv=SER_RX, ERRORS)
```

```
// ============================================================================
// Global variables
//

int delay_slots;



// ============================================================================
// Forward declarations
//

void Initialize(void);
void Delay_Adjust(void);

void Send_Serial(int data);
int  Recv_Serial(void);

void Service_RDA(void);



// ============================================================================
// Initialization procedure
//

void Initialize() {
   // Turn off interrupts
   disable_interrupts(GLOBAL);

   // Set tri-state for ports
   set_tris_a(TRIS_A);
   set_tris_c(TRIS_C);

   // Initialize port A for 3 ADC ports, Vdd internal reference
   setup_port_a(RA0_RA1_RA3_ANALOG);
   setup_adc(ADC_CLOCK_DIV_32);

   // Readjust number of delay slots if necessary
   Delay_Adjust();

   // Turn on interrupts
   enable_interrupts(INT_RDA);      // Enable serial data received interrupt
   enable_interrupts(GLOBAL);       // Activate interrupts
}


// Delay adjustment
void Delay_Adjust() {

   set_adc_channel(0);

   // Output version information
   if (ADJ_SEL) {
      printf("\n\r");
      printf("INTOUCH-3 DELAY SIMULATOR\n\r");
      printf("Revision ");
      printf(VERSION);
      printf("\n\r");
      printf("(C) 1998 MIT Media Lab\n\r");
      printf("Tangible Media Group\n\r");
   }

   // Adjustment loop
   while(ADJ_SEL) {
      delay_slots = read_adc();   // Sample ADC value

      // Can't have more delay slots than buffer capacity
      if (delay_slots > TOTAL_SIZE) delay_slots = TOTAL_SIZE;

      // Serial terminal output
```

```
      printf("delay slots: %u\n\r", delay_slots);

      delay_ms(500);
   }

   // Sample ADC to set number of delay slots
   delay_slots = read_adc();

}



// ============================================================================
// Serial communications procedures
//
// Send 8-bit data over the RS-232 serial interface

void Send_Serial(int data) {
   putc(data);
}


// Receive 8-bit data from the RS-232 serial interface
int Recv_Serial() {
   return (getc());
}



// ============================================================================
// Interrupt service routines (ISR)
//

#INT_RDA
// Read serial data when received
void Service_RDA() {
   static int count = 0;

   // Get received data
   Enqueue(Recv_Serial());

   // Wait delay_slots bytes then retransmit data
   if (count < delay_slots) {
      count++;
   } else {
      Send_Serial(Dequeue());
   }
}



// ============================================================================
// main
//

void main() {
   Initialize();
   while(1);
}
```

Data Collection Terminal

The following listing is a Java applet to facilitate the collection of data from the inTouch-3 system. The code assumes that two serial ports on the computer are

available. The listing below is again only the top-level file. A second library to implement threads to monitor the serial ports was also developed. The reader is referred to the accompanying CD-ROM for the complete listing.

```
// ============================================
// inTouch-3 data collection terminal
// ============================================

import java.awt.*;
import java.applet.*;
import Serialio.*;

public class MainApplet extends Applet {

    //{{DECLARE_CONTROLS
      java.awt.TextArea textArea1;
      java.awt.TextArea textArea2;
      //}}

      SerialReaderThread m_readThread1, m_readThread2;

      // default values
      String comport1 = "COM1", comport2 = "COM2";
      int bitrate1 = SerialConfig.BR_38400,
          bitrate2 = SerialConfig.BR_38400,
          size1    = SerialConfig.LN_8BITS,
          size2    = SerialConfig.LN_8BITS,
          stop1    = SerialConfig.ST_1BITS,
          stop2    = SerialConfig.ST_1BITS,
          parity1  = SerialConfig.PY_NONE,
          parity2  = SerialConfig.PY_NONE;


    public void init() {

        //{{INIT_CONTROLS
            setLayout(null);
            setSize(476,404);
            textArea1 = new java.awt.TextArea();
            textArea1.setBounds(12,12,420,192);
            add(textArea1);
            textArea2 = new java.awt.TextArea();
            textArea2.setBounds(12,216,420,172);
            add(textArea2);
            //}}
    }


    private int parseBitRate(String temp) {
        if (temp == null) {
            printMessage("Error parsing Bit Rate: " + temp);
            return bitrate1; // default
        }

        if (temp.equals("110"))         return SerialConfig.BR_110;
        else if (temp.equals("150"))    return SerialConfig.BR_150;
        else if (temp.equals("300"))    return SerialConfig.BR_300;
        else if (temp.equals("600"))    return SerialConfig.BR_600;
        else if (temp.equals("1200"))   return SerialConfig.BR_1200;
        else if (temp.equals("2400"))   return SerialConfig.BR_2400;
        else if (temp.equals("4800"))   return SerialConfig.BR_4800;
        else if (temp.equals("9600"))   return SerialConfig.BR_9600;
        else if (temp.equals("19200"))  return SerialConfig.BR_19200;
        else if (temp.equals("38400"))  return SerialConfig.BR_38400;
        else if (temp.equals("57600"))  return SerialConfig.BR_57600;
        else if (temp.equals("115200")) return SerialConfig.BR_115200;
        else {
```

```
        printMessage("Error parsing Bit Rate: " + temp);
        return bitrate1; // default
    }
}

private int parseSize(String temp) {
    if (temp == null) {
        printMessage("Error parsing size: " + temp);
        return size1; // default
    }

    if (temp.equals("8"))      return SerialConfig.LN_8BITS;
    else if (temp.equals("7")) return SerialConfig.LN_7BITS;
    else if (temp.equals("6")) return SerialConfig.LN_6BITS;
    else if (temp.equals("5")) return SerialConfig.LN_5BITS;
    else {
        printMessage("Error parsing size: " + temp);
        return size1; // default
    }
}

private int parseStopBit(String temp) {
    if (temp == null) {
        printMessage("Error parsing parity: " + temp);
        return parity1; // default
    }

    if (temp.equals("1"))      return SerialConfig.ST_1BITS;
    else if (temp.equals("2")) return SerialConfig.ST_2BITS;
    else {
        printMessage("Error parsing stop bit: " + temp);
        return stop1; // default
    }
}

private int parseParity(String temp) {
    if (temp == null) {
        printMessage("Error parsing parity: " + temp);
        return parity1; // default
    }

    if (temp.equals("none"))      return SerialConfig.PY_NONE;
    else if (temp.equals("odd"))  return SerialConfig.PY_ODD;
    else if (temp.equals("even")) return SerialConfig.PY_EVEN;
    else if (temp.equals("mark")) return SerialConfig.PY_MARK;
    else {
        printMessage("Error parsing parity: " + temp);
        return parity1; // default
    }
}

//private int (String temp)
//{
//}

public void start() {
    m_readThread1 = null;
    m_readThread2 = null;


    //Gets parameters from html
    String temp;
    comport1 = getParameter("comport1");
    comport2 = getParameter("comport2");

    temp     = getParameter("bitrate1");
    bitrate1 = parseBitRate(temp);

    temp     = getParameter("bitrate2");
    bitrate2 = parseBitRate(temp);
```

```
        temp  = getParameter("size1");
        size1 = parseSize(temp);

        temp  = getParameter("size2");
        size2 = parseSize(temp);

        temp  = getParameter("stop1");
        stop1 = parseStopBit(temp);

        temp  = getParameter("stop2");
        stop2 = parseStopBit(temp);

        temp    = getParameter("parity1");
        parity1 = parseParity(temp);

        temp    = getParameter("parity2");
        parity2 = parseParity(temp);

        printMessage(comport1, comport1 + " initializing...");
        m_readThread1 =
            new SerialReaderThread(this, comport1, bitrate1, size1, stop1, parity1,1000);

        printMessage(comport2, comport2 + " initializing...");
        m_readThread2 =
            new SerialReaderThread(this, comport2, bitrate2, size2, stop2, parity2,1000);
    }

    public void stop() {
        if ((m_readThread1 != null) && (m_readThread1.isAlive() )) {
            m_readThread1.stop();
        }

        if ((m_readThread2 != null) && (m_readThread2.isAlive() )) {
            m_readThread2.stop();
        }
    }

    public void destroy() {
        if ((m_readThread1 != null) && (m_readThread1.isAlive() )) {
            m_readThread1.closeFile();
            m_readThread1.stop();
        }

        if ((m_readThread2 != null) && (m_readThread2.isAlive() )) {
            m_readThread2.closeFile();
            m_readThread2.stop();
        }
    }

    public void printMessage(String devName, String message) {
        if (devName == comport1) {
            textArea1.append(message + "\n");
        } else {
            textArea2.append(message + "\n");
        }
     }

     public void printMessage(String message) {
        printMessage(comport1, message);
     }
}
```