Vindica Systems

Process Report for the Tactile Display

Project Team

Kevin Giroux
Stephen Hall
Leo Liting
Mike McFarland
Troy Tyler
Elaine Wong

Submitted to

Andrew Rawicz
Steve Whitmore
School of Engineering Science
Simon Fraser University

Date Issued

May 31, 2003

Revision

1.1 – May 28, 2003

Table of Contents

1 Introduction

In September of 2002 Vindica Systems (Vindica) launched a campaign into the design & development of a tactile interface system dubbed *TactiVision*. The driving force behind this undertaking was the need to facilitate improved access to electronic media for the visually impaired.

1.1 Tactile Displays

In concept, a tactile display (TD) is any device that can be used to convey information to its user through touch. In most instances, this is done by controlling the heights of a number of pins relative to one another within a matrix, so that the user can perceive surface-height variations with their hands and fingertips. This allows for the closest possible mapping from the 3D visual world to one of 2½D tactile. Other tactile displays utilize vibration or direct electrical stimulation to relay information, but these methods alter the acuity and feel of the perceived object making them inappropriate for many fundamental applications.

The TD concept is not a new one. There have been many advances in TD technology over the last 10 years, including the design and development of several commercially available (but expensive) refreshable Braille displays. ENSC340 itself has seen a refreshable Braille display project in recent history. What set Vindica's TD apart from its ancestry is its ability to produce analog pin-heights at a high-resolution. Vindica ranks well among the now dozens of organizations and research groups that have attempted to produce such a device, having successes that rival those of ventures which had budgets of over $4 million and project timelines in excess of 2 years. That any organization would invest that much time and money into such a device alludes to the large size of its potential market.

1.2 Market

Although no refreshable, analog tactile-display has been made commercially available to this day, the market potential for such a device has already been demonstrated. There are a number of refreshable Braille-type devices on the market which have been welcomed by the blind community in spite of their high costs. In addition to their obvious role as assistive devices for the blind, TDs have a number of other potential uses including:

- teletaction (feeling 3-dimensional objects / surfaces remotely) for pre-operative exploration of MRI or CT scan imagery by surgeons
- reusable electronics test-beds for rapid printed-circuit testing (bed of nails)

There are undoubtedly also a number of other useful applications that remain undiscovered thus far. Therefore, the market potential for refreshable, analog TDs exists even if the technology is expensive; however, if the design uses technology that is affordable to the average consumer, we feel that the resulting market for such a device would be enormous and could easily be swept by the first supplier.

2 TactiVision

Vindica System's '*TactiVision*' is a proof-of-concept, refreshable, analog TD system whose primary intended use is as an assistive device for the visually impaired. An obvious change to any production version would be the development of invisible software to relay data from other programs to the TD, or the development of a controller that plugs straight into the VGA port, replacing the monitor altogether.

2.1 System Overview

Vindica's TD system consists of a graphical user interface (GUI), controller, and the tactile user interface (TUI). The GUI accepts user inputs in the form of an image which can be manipulated through the software in a manner similar to that of the common paint program. These inputs are then transmitted from the PC to the controller from which motor commands are sent to the TUI actuators. The TUI then displays 2½ dimensional contours by adjusting pin-heights so that they represent what is shown on the GUI, allowing one to visualize through touch. A graphical representation of the system overview is shown in Figure 1.1 below.

Figure 1.1 - System Overview (GUI ⇨ controller ⇨ TUI)

2.2 Graphical User Interface

The GUI is a Win32 based application written in Microsoft Visual C++ 6.0 using Microsoft Foundation Classes (MFC). A complete, commented copy of the code has been included in Appendix A. The program consists of a single dialog-style window with 7 functions that the user can perform:

- *Open* – activates standard Windows® File>Open dialog box where user can browse and select .tvi image to be loaded into editor
- *Palette* – allows user to select current brush color from a 256 degree grayscale gradient [black = low : white = high]
- *Get Color*– current color is updated with color of next pixel clicked in editor window (similar to 'eyedropper' tool in other image software)
- *Paint* – left-click in editor window applies current brush color to selected pixels (similar to 'paintbrush' tool in other image software)
- *Pan* – allows user to highlight and select a 19 -pixel honey-comb region of the image to display on the 19 -pin tactile interface
- *Update* – initiates transmission of selected 19 -pixel data to controller
- *Save* – activates standard Windows® File>Save dialog box where user is prompted to enter file name to save image in editor

Each pixel in the selected region has a brightness value between 0 (black) and 255 (white). This mapping is inverted with respect to the TUI actuator commands which use white (completely off : high) and black (completely on : low), so the number is subtracted from 255 to get the desired output. The software enables the computer's COM1 serial port and controls asynchronous transmission of pin-height data directly with the PIC's built-in hardware SPI, via UART protocol. The PIC stores the data so that each image need only be updated once.

2.3 Controller

The brain of the control hardware is a Microchip® PIC16F628 flash MCU which was programmed using MPLab IDE and a *PICPlus* programmer. A complete, commented copy of the code has been included in Appendix B. The on-chip SPI receives data from the PC after it has passed through a logic inverter.

The PIC receives one byte at a time, saving the values in memory as they arrive. Once all 19 values have been received, they are sequentially latched into 8-bit digital-to-analog-converters (DAC) using 2x 4-to-16 active-low line-decoders for addressing and parallel 5-bit input lines. Only the most significant 5 bits of each pin-height value are latched, with the remaining 3 leads being tied to ground. This had to be done due to microcontroller output constraints, yet it still allows 32 unique pin-heights to be displayed (increments of approx. 70 μm deflection).

The DAC outputs are buffered using parallel 200 mA high-power op-amps which connect directly to a 20-pin connector that connects the controller to the TUI. It should be noted that this design has extremely low tolerance for variations in the resistances of the muscle wire actuators (which occurs quite regularly). The control system schematics for a single pin are shown in Figure 2.1.

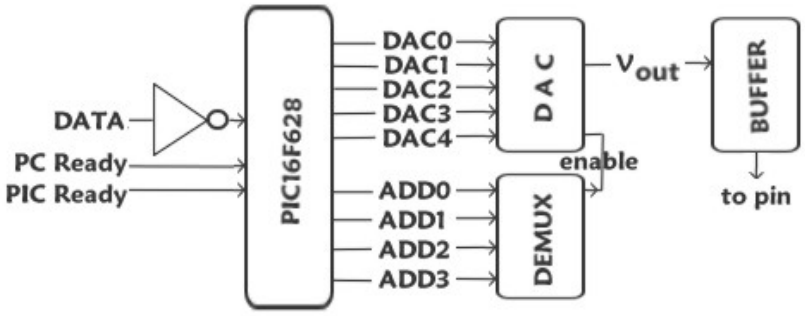


Figure 2.1 – High-Level Control System Schematic

The design shown in Figure 2.1 deviates significantly from the design specifications previously released. The original design used capacitors and transistors instead of DACs and op-amps, and would have been significantly less expensive than the new design; however, the original design was deemed to be flawed late into the development process and was subsequently replaced by that shown in Figure 2.1.

The signals DAC0 - DAC4 are available at the inputs of each DAC, and the addressing lines ADD0 - ADD3 select which of the DACs will latch the current value. When a DAC receives an active low from the decoder, the voltage at its output is updated and reflected at the output of the voltage buffer (op-amps).

This design uses simple R2R ladder networks and transistors so it could be integrated onto an IC, however the number of pins required on the package would increase in direct proportion with the number of pins on the tactile display.

2.4 Tactile User Interface

The TUI is a 19-pin array of hexagonal pins with shape memory alloy (SMA) actuators. When heated by applying a potential difference across the wire, SMA contracts pulling its associated pin downwards. Compression springs provide the restoring force which pushes the pins back to their original heights as they cool. The controller supplies each of the pins with its own designated voltage, precisely regulating their heights. Although the mechanics behind the pin design are simple in concept, they are difficult to implement with any precision by hand. Diagrams of the TD design and construction process have been included in *Appendix C*.

2.5 Problems Encountered

The system was initially designed to conform to target cost constraint of $0.50 per pin, however the first design had to be abandoned because it involved the construction of and analog latch for each pin, a task that proved to be much more difficult than expected. This necessitated a design overhaul late into the project.

Most of the problems encountered in the project were related to construction of the TUI. The SMA used to actuate the pins was very difficult to work with for all of the following reasons:

- heat sensitive (80° – 90° activation range)
  - soldering burns out wire
  - hysteresis in temperature vs length characteristic
- thinner than human hair (100 μm diameter)
  - difficult to work with by hand
  - knots untie themselves when wire is heated
- non-stick, non-malleable alloy
  - difficult to fasten
  - force required to clamp often fractures wire
- non-uniform resistivity (approx 1 Ω per cm)
  - resistance varies along length & with length
  - small voltages (1.2 volts – 1.7 volts) burn out wires

A conservative estimate of the total amount of time spent designing, constructing, testing, redesigning and reconstructing the pins is 400 hours. Pins that worked would do so for a few contraction-lengthening cycles before either sliding through the fastening crimp, or burning out. Some of this was due to what was observed to be an inconsistency between resistances of equal lengths of the wire (two 5 cm

pieces measured between 3.4 Ω and 54.1 Ω). As a result, Vindica's final product only had two contracting pins at the time of demonstration, though several of the pins worked at some time prior to that point.

In addition to these project related issues, Vindica Systems suffered the loss of 3 of its members well prior to project completion. Those who remained spent weeks (including Christmas Eve!) trying to debug firmware authored by those who had just left, without success. The firmware had to be rewritten from scratch and undergo later modification when the control design was discovered to be flawed.

3 Budget

Vindica Systems was well under budget at the time of integration of its first design. The design change drove the final costs up significantly, and was a failure of concept in many ways. If the cost of pin-assembly labor were factored into the cost of the system, the final price-tag would be thousands of dollars more. Of the $1200.00 allocated for the project, $896.86 was spent. A summary of those expenses is shown below:

$115.34 – shape memory alloy sample kit (3m)
$115.34 – 100 μm shape memory alloy (5m)
$ 16.96 – 280 mA continuous transistors (50)
$ 36.09 – PIC16F628 microcontrollers (3)
$138.59 – crimps, connectors, sockets, springs, enclosure, switch, standoffs
$ 56.27 – thermal paste, small fan, heat sinks
$ 65.76 – power supply, connector, fuses
$210.06 – 200 mA high-power op-amps (40)
$107.67 – 8-bit digital-to-analog converters (20)
$ 34.78 – 4-to-16 active-low line decoders (3)

$896.86 – total

The final cost varies so significantly from our predicted budget requirements because we initially expected to pay as much as $400.00 for SMA but only paid $230.00. We were also able to obtain several free samples to offset some of the cost of our initial design, and in spite of the unforeseen design change we were able to offset the contingency cost of our second design against savings of the first.

4 Timeline

To say the very least, Vindica's actual project timeline strayed erratically from the timeline proposed in September. Some of the deviation is credited to the late start the founding members made, meeting and brainstorming for the first time only days before the first ENSC305 class. Because the team was small it became the leading candidate for an insertion of 3 foreign exchange students. The addition of 3 more minds and 6 more helping hands was received as a blessing at first, but as time wore on it became clear that the lopsided division of tasks within the group was leading to significant delays. The effect of group dynamics on this project is discussed further in Section5.

A graphical timeline has been included in *Appendix D* as it is too large for this page. The most significant deviations from our predicted schedule were the late development of the initial firmware, which could not be debugged before its authors returned home, and the resulting delay in system integration which revealed the design error. Vindica's remaining members were also engaged in full course loads in the spring semester, leaving even less time to work on the project than had been the case in the fall.

5 Group Dynamics

As mentioned in the previous section, group dynamics played an important role in the timeline of the project. Two of the exchange students had very little practical hands-on experience with electronics test equipment or hardware. The third excelled in each of the other courses taken while visiting, yet surprisingly and inexplicably failed to contribute much to either of ENSC305/340.

Whether the assumption of a more dominant leadership role by any of the group members would have helped resolve this issue is uncertain. Only 2 pairs of members knew each other prior to formation of the group, and the storming stage went surprisingly well and quickly considering. Norming took considerably longer than it should have in retrospect, due to the unavailability of certain members as well as to differing work ethics and habits within the group.

A general lack of team cohesion in the early stages of the project meant that the initiation of work in each facet of the design repeatedly fell to the same person. Vindica's leading project contributors were also guilty of spending more time

producing documentation than they should have, given the skewed division of tasks within the group. In spite of the possible drawbacks to working on a project of this magnitude with people you don't know, some of which were just mentioned, most of Vindica's members would agree that the experience reflects the real-world work-place. In many respects we've learned more about team dynamics and project management than we have about electronics throughout the course of this project, and the experience has been invaluable.

6 Changes to Make

If Vindica Systems were to tempt fate once more and try to redevelop the TD system from scratch, there are a few changes that would occur. Most importantly, the actuating scheme would be more closely scrutinized. Shape memory alloys are probably not the best suited material for this particular application, though they will work. One other actuation method that Vindica would have liked to have had more time to investigate last September was hydraulics. Though we're not mechanically inclined, it has as much promise as any other mechanism. If Vindica decided to continue using SMA, custom pre-cut and pre-crimped lengths of wire are now offered by our previous muscle-wire supplier, and might eliminate some of the problems encountered with the interface assembly process.

Another change to make is the implementation of the control hardware on a printed circuit board (PCB). In spite of our confidence in Kevin's superhuman soldering skills, 700 manual connections virtually eliminate any hope of successfully debugging shorts or improper circuitry.

Lastly, Vindica's design change came about so late in the process because we were slow to exploit the resources and knowledge base available to us. Had we seen Lucky sooner, or consulted with Dr Rawicz regarding some of our difficulties with SMA we may well have been finished on schedule.

One thing that we wouldn't change is the complexity and challenge of the undertaking. There was much learned due to the struggle, about project management and electronics, but also about ourselves and each other. Had we chosen a simpler ENSC340 project we may have finished without learning a thing.

7 Individual Contributions

The following is a summary of each of Vindica's group members' contributions to the project. Each is a personal account of what was done, with the exception of the exchange students' contributions, which have been recorded in their absence.

Kevin Giroux – When 340 began, I was looking forward to the opportunity to do some hands on engineering work. With my experience in soldering and building electronic devices, such as an audio power meter and multimedia amplifier, my efforts were directed towards building the circuitry, the enclosure, and the pins themselves. I learned through hand soldering around 700 connections on a generic proto board that fabricating a printed circuit board or even just using wire wraps would have been a more efficient and reliable technique. I also learned that while plenty of heat is essential for a good solder joint, it is not the best technique to use when working with a substance that changes its properties as its temperature varies, such as the nickel-titanium alloy we were using to actuate the pins.

Aside from the frustrating process of manufacturing pins that did not work, one of the most tedious parts of this project was building the case that would contain all of the pins. Due to the complexity of the shape of our pin array, we had no other means at our disposal besides hand filing it. This was an exercise in patience as much as anything else, as one careless file stroke would result in having to start over from scratch on a new case.

In addition to the practical experience gained in building the tactivision, I also gained experience, through 305, in writing technical documents. While it was disappointing not to see 19 functional pins at the end of the project, all the effort that went into getting there was a very valuable learning experience.

Stephen Hall – Steve took the initiative to purchase a muscle-wire sample kit and test the SMA's strength to see if it met project requirements. He also proof-read Vindica's project documentation and helped draw up funding requests. Though his contributions to the project were probably less than what they could have been during his time here, in his defense he was taking several of the more difficult courses in the SFU engineering program concurrently with ENSC305/340.

Leo Liting & Elaine Wong – Leo Liting and Elaine Wong, our team members from Singapore, were inseparable and although we are unaware of their individual contributions, we do know what they added as a team. Leo and Elaine were responsible for selecting and sourcing a microcontroller suitable for the project. They chose the Microchip PIC16F628 because it met all of our requirements and was relatively inexpensive and easy to program. They wrote the first version of the firmware for our first design, but returned to Singapore before we were able to test it. Nonetheless, some of their contributions snuck their way into the final version of the firmware, and controlled Vindica's TD at the demo.

Mike McFarland – In addition to working on all of the ENSC305 documentation and writing the presentation, I was responsible for the development & debug of the software and the firmware following Leo & Elaine's departure. As CFO, I sought project funding and sourced, ordered and financed all of the project components. I helped design, bread-board and test several control hardware schemes and, like the others, spent countless hours constructing pins and becoming frustrated by muscle wire. My contributions involved every aspect of the project.

Troy Tyler – At the inception of the project I spent many hours researching ways to actuate the pins, and to generally make the project happen. I worked extensively with Mike both in the research and in producing the Project Proposal. Mike and I also had a large part in all documentation. In the early months while Mike was programming the software interface and I was advising, I built the company website www.vindica.ca. During the project development stages I spent many hours in the lab prototyping ways to actuate muscle wire. Mike and I built the initial design circuitry and ultimately the final design circuitry. I also had fun soldering the first of those 700+ connections on the final control board. I assisted with the firmware as well, though I did manage to break a few pins on a few chips. We then integrated the circuitry with the software. At this point we had to finish the pin assembly as its state at the time was unusable. I spent many, many hours filing, cutting, tying, crimping, building and re-building pins and the enclosure. When it was all done we had 4 pins actuating! But alas, by the time we demoed we were down to 1 or 2. I made the slides for the pin enclosure because I got to have cool graphics to make, and was fortunate to be the one to explain all the work involved in making the pins. I found the group members that I worked with on those long days very fun to work with, which is probably why we are all still here today.

8 Conclusion

8.1 Future Plans

Despite our sense that tremendous potential exists for tactile display technology, we have no immediate plans to pursue its development any further. The device is at a stage where it could be made to work with only minor changes however the design falls short of its original component cost target and has high assembly costs.

8.2 Recommendations

To management or any design teams considering the pursuit of a solution to this problem in the future, we would strongly recommend against the use of shape memory alloy in its design (at least SMA of the 'muscle wire' variety). We felt that the project was challenging in all aspects of electronics circuitry, software and firmware design, and would recommend projects of similar magnitude to future students. We truly appreciate Steve and Andrew's willingness to allow us to see the project through to completion and each of us enjoyed ENSC305/340.

Appendix A

GUI Source Code

```
// Written by Mike McFarland © Vindica Systems 2002
// TactiVision.h : main header file for the TACTIVISION application
#include "resource.h"

class CTactiVisionApp : public CWinApp
{
public:
        CTactiVisionApp();

        //{{AFX_VIRTUAL(CTactiVisionApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL
};

// Written by Mike McFarland © Vindica Systems 2002
// TactiVision.cpp : Defines the class behaviors for the application.

#include "stdafx.h"
#include "TactiVision.h"
#include "TactiVisionDlg.h"

CTactiVisionApp theApp;

BOOL CTactiVisionApp::InitInstance()
{
#ifdef _AFXDLL
        Enable3dControls();                     // Call this when using MFC in a shared DLL
#else
        Enable3dControlsStatic();   // Call this when linking to MFC statically
#endif
        CTactiVisionDlg dlg;
        m_pMainWnd = &dlg;
        int nResponse = dlg.DoModal();
        return FALSE;
}

#include "stdafx.h"
#include "TactiVision.h"
#include "TactiVisionDlg.h"
#include "windows.h"
#include <math.h>

#define zoomLeft            276
#define zoomTop              97
```

```
#define gradLeft        250
#define gradTop         118

#define openLeft        323
#define openTop          62
#define openRight       391
#define openBottom       79

#define saveLeft        468
#define saveTop          62
#define saveRight       527
#define saveBottom       79

#define areaLeft         62
#define areaTop         262
#define areaRight       205
#define areaBottom      279

#define colorLeft        90
#define colorTop        228
#define colorRight      205
#define colorBottom     241

#define updateLeft      115
#define updateTop       304
#define updateRight     205
#define updateBottom    321

Hexagon::Hexagon()
{
        vertex[0].x = vertex[0].y = 0,
        vertex[1].x = vertex[1].y = 0,
        vertex[2].x = vertex[2].y = 0,
        vertex[3].x = vertex[3].y = 0,
        vertex[4].x = vertex[4].y = 0,
        vertex[5].x = vertex[5].y = 0,
        hexcolor    = 0x00FFFFFF;
}

void Hexagon::setHexagon(int xcoord, int ycoord, COLORREF color)
{
        hexcolor = color;
        vertex[0].x = vertex[5].x = xcoord;
        vertex[0].y = vertex[2].y = ycoord;
        vertex[1].x = vertex[4].x = xcoord + 5;
        vertex[1].y = ycoord - 3; vertex[4].y = ycoord + 9;
        vertex[2].x = vertex[3].x = xcoord + 10;
        vertex[3].y = vertex[5].y = ycoord + 6;
        vertex[6].x = xcoord + 5;
```

```
	vertex[6].y = ycoord + 3;
}
void CTactiVisionDlg::SetHexagons()
{
	int xRef = zoomLeft; int yRef = zoomTop; int xOffset = 5;
	int deltaX = 10; int deltaY = 9; int nexthex = 0;

	for (int row=0; row<31; row++)
		if (row%2 == 0)
			for (int col=0; col<30; col++)
			{zoomhex[nexthex].setHexagon(xRef+col*deltaX,
			 yRef+row*deltaY, (COLORREF) 0x00FFFFFF);
			 nexthex++; }
		else
			for (int col=0; col<29; col++)
			{zoomhex[nexthex].setHexagon(xRef+col*deltaX+
			 xOffset,yRef+row*deltaY, (COLORREF) 0x00FFFFFF);
			 nexthex++; }

	CPen outlinePen(PS_SOLID,1,(COLORREF) 0xE1B982);
	screen->SelectObject(&outlinePen);

	for (int thishex=0; thishex<915; thishex++)
	{
		screen->MoveTo(zoomhex[thishex].getVertex1().x,zoomhex[thishex].getVertex1().y);
		screen->LineTo(zoomhex[thishex].getVertex2().x,zoomhex[thishex].getVertex2().y);
		screen->LineTo(zoomhex[thishex].getVertex3().x,zoomhex[thishex].getVertex3().y);
		screen->LineTo(zoomhex[thishex].getVertex4().x,zoomhex[thishex].getVertex4().y);
		screen->LineTo(zoomhex[thishex].getVertex5().x,zoomhex[thishex].getVertex5().y);
		screen->LineTo(zoomhex[thishex].getVertex6().x,zoomhex[thishex].getVertex6().y);
		screen->LineTo(zoomhex[thishex].getVertex1().x,zoomhex[thishex].getVertex1().y);
	}
	outlinePen.DeleteObject();
}

void Hexagon::FillPoly(CDC* screen)
{
	CPen outlinePen(PS_SOLID,1,(COLORREF) 0xE1B982);
	screen->SelectObject(&outlinePen);
	CBrush hexFillBrush(this->getColor());
	screen->SelectObject(&hexFillBrush);
	screen->Polygon(vertex,6);
	outlinePen.DeleteObject();
	hexFillBrush.DeleteObject();
}

CTactiVisionDlg::CTactiVisionDlg(CWnd* pParent)
	: CDialog(CTactiVisionDlg::IDD, pParent)
{
	tvIcon  = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
```

```
	keyStop = ::LoadAccelerators(AfxGetInstanceHandle(), MAKEINTRESOURCE(LimitKeypress));
}

void CTactiVisionDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CTactiVisionDlg)
	//}}AFX_DATA_MAP
}

BOOL CTactiVisionDlg::PreTranslateMessage(MSG* incoming)
{
	if (keyStop != NULL)
		if (::TranslateAccelerator(m_hWnd, keyStop, incoming)) return TRUE;
	return CDialog::PreTranslateMessage(incoming);
}

BEGIN_MESSAGE_MAP(CTactiVisionDlg, CDialog)
	//{{AFX_MSG_MAP(CTactiVisionDlg)
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_WM_LBUTTONDOWN()
	ON_WM_MOUSEMOVE()
	ON_WM_LBUTTONUP()
	ON_BN_CLICKED(LoadButton, OnLoadButton)
	ON_BN_CLICKED(PanButton, OnPanButton)
	ON_WM_CONTEXTMENU()
	ON_BN_CLICKED(SaveButton, OnSaveButton)
	ON_BN_CLICKED(ColorButton, OnGetColor)
	ON_BN_CLICKED(UpdateButton, OnUpdateButton)
	ON_WM_RBUTTONDOWN()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CTactiVisionDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	SetIcon(tvIcon, TRUE);			// Set big icon
	SetIcon(tvIcon, FALSE);			// Set small icon

	screen		= this->GetDC();
	currColor	= (COLORREF) 0x00FFFFFF;
	lastFrame	= 100;
	lastColor	= 100;
	cursorHidden	= FALSE;
	inPanMode	= FALSE;
	inGetColorMode = FALSE;
	inDrawArea	= FALSE;
	hasPanned		= FALSE;
	hasColor	= FALSE;
```

```
	SetHexagons();
	currColor = (COLORREF) 0x00000000;
	colorHex.setHexagon(gradLeft + 2, gradTop - 14, (COLORREF) 0x00000000);
	GUILayoutMemory.CreateCompatibleDC(screen);

	//Load Application Backdrop into Memory
	CBitmap guiBMP;
	guiBMP.LoadBitmap(BackDrop);
	GUILayoutMemory.SelectObject(guiBMP);
	guiBMP.Detach();
	guiBMP.DeleteObject();

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CTactiVisionDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting
		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, tvIcon);
	}
	else
	{

		screen->BitBlt(0,0,600,400,&GUILayoutMemory,0,0,SRCCOPY);
		CPen framePen(PS_SOLID,1,(COLORREF) 0x000000FF);
		if (hasPanned) FrameHandle(lastFrame, &framePen);
		framePen.DeleteObject();

		colorHex.FillPoly(screen);
		for (int index=0; index<915; index++)
			zoomhex[index].FillPoly(screen);

		// Pass windows message to CDialog class OnPaint handler
		CDialog::OnPaint();
	}
}
```

```
HCURSOR CTactiVisionDlg::OnQueryDragIcon()
{ return (HCURSOR) tvIcon; }

int CTactiVisionDlg::GetHexagon(CPoint point)
{
	int   x,y;
	int   currentMinHex = 0;
	float hexDistance  = 0;
	float minDistance = 500;

	for (int index=0; index<915; index++)
	{	x = abs(zoomhex[index].getCenter().x - point.x);
		y = abs(zoomhex[index].getCenter().y - point.y);
		hexDistance = (float) sqrt((x*x)+(y*y));
		if (hexDistance < minDistance)
		{	currentMinHex = index;
			minDistance = hexDistance;	}	}
	return currentMinHex;
}

void CTactiVisionDlg::GetDrawing(CPoint point)
{
	CPen framePen(PS_SOLID,1,(COLORREF) 0x000000FF);
	int drawnOn = GetHexagon(point);
	{	zoomhex[drawnOn].setColor(currColor);
		zoomhex[drawnOn].FillPoly(screen);
		if (hasPanned) FrameHandle(lastFrame, &framePen);	}
	framePen.DeleteObject();
}

void CTactiVisionDlg::OnLButtonDown(UINT nFlags, CPoint point)
{

	if (zoomhex[0].getVertex1().x <= point.x && zoomhex[0].getVertex2().y <= point.y
	&&  zoomhex[914].getVertex4().x >= point.x && zoomhex[914].getVertex5().y >= point.y)
		if(!inPanMode && !inGetColorMode)
		{
			inDrawArea = TRUE;
			GetDrawing(point);
		}
		if(inPanMode)
		{
			inPanMode = FALSE;
			::ShowCursor(TRUE);
			::ClipCursor(NULL);
			cursorHidden = FALSE;
		}
		if(inGetColorMode)
```

```
		{
			inGetColorMode = FALSE;
			CPen frameColor (PS_SOLID, 1,(COLORREF) 0xE1B982);
			currColor = zoomhex[GetHexagon(point)].getColor();
			GetColorHandle(GetHexagon(point), &frameColor);
			::ShowCursor(TRUE);
			::ClipCursor(NULL);
			cursorHidden = FALSE;
			frameColor.DeleteObject();
		}

	if (point.x >= gradLeft && point.x <= gradLeft + 10
	&&  point.y >= gradTop  && point.y <= gradTop + 255)
	{
		int pixelToTake = point.y - gradTop;
		if (pixelToTake < 0 ) pixelToTake = 0;
		if (pixelToTake > 255 ) pixelToTake = 255;
		pixelToTake = 255 - pixelToTake;
		currColor = ((COLORREF) ((pixelToTake << 16L) + (pixelToTake << 8L) +
			(pixelToTake)));
	}

	colorHex.setHexagon(gradLeft + 2, gradTop - 14, currColor);
	colorHex.FillPoly(screen);

	if (point.x >= openLeft && point.x <= openRight &&
		point.y >= openTop  && point.y <= openBottom)
		OnLoadButton();

	if (point.x >= areaLeft && point.x <= areaRight &&
		point.y >= areaTop  && point.y <= areaBottom)
		OnPanButton();

	if (point.x >= saveLeft && point.x <= saveRight &&
		point.y >= saveTop  && point.y <= saveBottom)
		OnSaveButton();

	if (point.x >= colorLeft && point.x <= colorRight &&
		point.y >= colorTop  && point.y <= colorBottom)
		OnGetColor();

	if (point.x >= updateLeft && point.x <= updateRight &&
		point.y >= updateTop  && point.y <= updateBottom)
		OnUpdateButton();

	CDialog::OnLButtonDown(nFlags, point);
}
```

```
void CTactiVisionDlg::FrameHandle(int centerHex, CPen* currentPen)
{
	screen->SelectObject(currentPen);

	screen->MoveTo(zoomhex[centerHex- 2].getVertex1().x, zoomhex[centerHex- 2].getVertex1().y);
	screen->LineTo(zoomhex[centerHex- 2].getVertex6().x, zoomhex[centerHex- 2].getVertex6().y);
	screen->LineTo(zoomhex[centerHex- 2].getVertex5().x, zoomhex[centerHex- 2].getVertex5().y);
	screen->LineTo(zoomhex[centerHex+28].getVertex6().x, zoomhex[centerHex+28].getVertex6().y);
	screen->LineTo(zoomhex[centerHex+28].getVertex5().x, zoomhex[centerHex+28].getVertex5().y);
	screen->LineTo(zoomhex[centerHex+58].getVertex6().x, zoomhex[centerHex+58].getVertex6().y);
	screen->LineTo(zoomhex[centerHex+58].getVertex5().x, zoomhex[centerHex+58].getVertex5().y);
	screen->LineTo(zoomhex[centerHex+58].getVertex4().x, zoomhex[centerHex+58].getVertex4().y);
	screen->LineTo(zoomhex[centerHex+59].getVertex5().x, zoomhex[centerHex+59].getVertex5().y);
	screen->LineTo(zoomhex[centerHex+59].getVertex4().x, zoomhex[centerHex+59].getVertex4().y);
	screen->LineTo(zoomhex[centerHex+60].getVertex5().x, zoomhex[centerHex+60].getVertex5().y);
	screen->LineTo(zoomhex[centerHex+60].getVertex4().x, zoomhex[centerHex+60].getVertex4().y);
	screen->LineTo(zoomhex[centerHex+60].getVertex3().x, zoomhex[centerHex+60].getVertex3().y);
	screen->LineTo(zoomhex[centerHex+31].getVertex4().x, zoomhex[centerHex+31].getVertex4().y);
	screen->LineTo(zoomhex[centerHex+31].getVertex3().x, zoomhex[centerHex+31].getVertex3().y);
	screen->LineTo(zoomhex[centerHex+ 2].getVertex4().x, zoomhex[centerHex+ 2].getVertex4().y);
	screen->LineTo(zoomhex[centerHex+ 2].getVertex3().x, zoomhex[centerHex+ 2].getVertex3().y);
	screen->LineTo(zoomhex[centerHex+ 2].getVertex2().x, zoomhex[centerHex+ 2].getVertex2().y);
	screen->LineTo(zoomhex[centerHex-28].getVertex3().x, zoomhex[centerHex-28].getVertex3().y);
	screen->LineTo(zoomhex[centerHex-28].getVertex2().x, zoomhex[centerHex-28].getVertex2().y);
	screen->LineTo(zoomhex[centerHex-58].getVertex3().x, zoomhex[centerHex-58].getVertex3().y);
	screen->LineTo(zoomhex[centerHex-58].getVertex2().x, zoomhex[centerHex-58].getVertex2().y);
	screen->LineTo(zoomhex[centerHex-58].getVertex1().x, zoomhex[centerHex-58].getVertex1().y);
	screen->LineTo(zoomhex[centerHex-59].getVertex2().x, zoomhex[centerHex-59].getVertex2().y);
	screen->LineTo(zoomhex[centerHex-59].getVertex1().x, zoomhex[centerHex-59].getVertex1().y);
	screen->LineTo(zoomhex[centerHex-60].getVertex2().x, zoomhex[centerHex-60].getVertex2().y);
	screen->LineTo(zoomhex[centerHex-60].getVertex1().x, zoomhex[centerHex-60].getVertex1().y);
	screen->LineTo(zoomhex[centerHex-60].getVertex6().x, zoomhex[centerHex-60].getVertex6().y);
	screen->LineTo(zoomhex[centerHex-31].getVertex1().x, zoomhex[centerHex-31].getVertex1().y);
	screen->LineTo(zoomhex[centerHex-31].getVertex6().x, zoomhex[centerHex-31].getVertex6().y);
	screen->LineTo(zoomhex[centerHex- 2].getVertex1().x, zoomhex[centerHex- 2].getVertex1().y);
}

void CTactiVisionDlg::PanDrawArea(CPoint point)
{
	CPen frameColor (PS_SOLID, 1,(COLORREF) 0xE1B982);
	CPen selectColor (PS_SOLID, 1,(COLORREF) 0x0000FF);

	int centerHex = GetHexagon(point);
	FrameHandle(lastFrame, &frameColor);
	FrameHandle(centerHex, &selectColor);
	lastFrame = centerHex;

	frameColor.DeleteObject();
```

```
	selectColor.DeleteObject();
}

void CTactiVisionDlg::GetColorHandle(int centerHex, CPen* currentPen)
{
	screen->SelectObject(currentPen);

	screen->MoveTo(zoomhex[centerHex].getVertex1().x, zoomhex[centerHex].getVertex1().y);
	screen->LineTo(zoomhex[centerHex].getVertex2().x, zoomhex[centerHex].getVertex2().y);
	screen->LineTo(zoomhex[centerHex].getVertex3().x, zoomhex[centerHex].getVertex3().y);
	screen->LineTo(zoomhex[centerHex].getVertex4().x, zoomhex[centerHex].getVertex4().y);
	screen->LineTo(zoomhex[centerHex].getVertex5().x, zoomhex[centerHex].getVertex5().y);
	screen->LineTo(zoomhex[centerHex].getVertex6().x, zoomhex[centerHex].getVertex6().y);
	screen->LineTo(zoomhex[centerHex].getVertex1().x, zoomhex[centerHex].getVertex1().y);
}

void CTactiVisionDlg::GetColorDrawArea(CPoint point)
{
	CPen frameColor (PS_SOLID, 1,(COLORREF) 0xE1B982);
	CPen selectColor (PS_SOLID, 1,(COLORREF) 0x0000FF);
	int centerHex = GetHexagon(point);
	GetColorHandle(lastColor, &frameColor);
	GetColorHandle(centerHex, &selectColor);
	lastColor = centerHex;
	frameColor.DeleteObject();
	selectColor.DeleteObject();
}

void CTactiVisionDlg::OnMouseMove(UINT nFlags, CPoint point)
{
	if (inPanMode)
	{
		if (cursorHidden && (point.x < zoomhex[0].getVertex1().x || point.y <
		zoomhex[0].getVertex2().y
		|| point.x > zoomhex[914].getVertex3().x || point.y > zoomhex[914].getVertex5().y))
		{ ::ShowCursor(TRUE); cursorHidden = FALSE; }

		if (!cursorHidden && (point.x >= zoomhex[0].getVertex1().x && point.y >=
		zoomhex[0].getVertex2().y
		&& point.x <= zoomhex[914].getVertex3().x && point.y <=
		zoomhex[914].getVertex5().y))
		{ ::ShowCursor(FALSE); cursorHidden = TRUE;	}
		PanDrawArea(point);
	}

	if (inGetColorMode)
	{
		if (cursorHidden && (point.x < zoomhex[0].getVertex1().x || point.y <
		zoomhex[0].getVertex2().y
		||point.x >zoomhex[914].getVertex3().x||point.y > zoomhex[914].getVertex5().y))
```

```
                { ::ShowCursor(TRUE); cursorHidden = FALSE; }

                if (!cursorHidden && (point.x >= zoomhex[0].getVertex1().x && point.y >=
                zoomhex[0].getVertex2().y
                        && point.x <= zoomhex[914].getVertex3().x && point.y <=
                zoomhex[914].getVertex5().y))
                { ::ShowCursor(FALSE); cursorHidden = TRUE;          }
                GetColorDrawArea(point);
        }
        if (inDrawArea) GetDrawing(point);
        CDialog::OnMouseMove(nFlags, point);
}

void CTactiVisionDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
        inDrawArea = FALSE;
        CDialog::OnLButtonUp(nFlags, point);
}

void CTactiVisionDlg::OnLoadButton()
{
        CString fileName(_T(""));
        TCHAR filters[] = _T("TactiVision Image (.tvi)|*.tvi||");
        CFileDialog loadTVI (TRUE, _T("tvi"), _T("*.tvi"), OFN_FILEMUSTEXIST
                        | OFN_HIDEREADONLY, filters);
        if (loadTVI.DoModal() == IDOK) fileName = loadTVI.GetPathName();
        if (fileName != _T(""))
        {
                CFile tviFile(fileName, CFile::modeRead);
                COLORREF loadColor;
                CArchive tviArchive(&tviFile, CArchive::load);//, 8192);

                for (int index=0; index<915; index++)
                {
                        tviArchive >> loadColor;
                        zoomhex[index].setColor(loadColor);
                }

        tviArchive.Close();
        tviFile.Close();
        }
        OnPaint();
}

void CTactiVisionDlg::OnSaveButton()
{
        CString fileName;
        fileName.Empty();

        TCHAR filters[] = _T("TactiVision Image (.tvi)|*.tvi||");
```

```
		CFileDialog saveTVI (FALSE, _T("tvi"), _T("*.tvi"), OFN_OVERWRITEPROMPT, filters);
		if (saveTVI.DoModal() == IDOK) fileName = saveTVI.GetPathName();
		if (!fileName.IsEmpty())
		{
			CFile tviFile(fileName, CFile::modeReadWrite | CFile::modeCreate);
			CArchive tviArchive(&tviFile, CArchive::store);
			for (int index=0; index<915; index++) tviArchive << zoomhex[index].getColor();

		tviArchive.Close();
		tviFile.Close();
		}
}

void CTactiVisionDlg::OnPanButton()
{
		CRect panWindow;
		this->GetWindowRect(&panWindow);
		hasPanned = TRUE;
		CPoint topLeftClipPoint (zoomhex[3].getVertex1().x + 2 + panWindow.left,
					zoomhex[150].getVertex2().y + 2 + panWindow.top);
		CPoint bottomRightClipPoint (zoomhex[912].getVertex3().x - 2 + panWindow.left,
					zoomhex[890].getVertex5().y + 0 + panWindow.top);
		CRect clipRegion (topLeftClipPoint, bottomRightClipPoint);
		::SetCursorPos(((topLeftClipPoint.x + bottomRightClipPoint.x)/2), ((topLeftClipPoint.y +
					bottomRightClipPoint.y)/2));
		::ClipCursor(clipRegion);
		inPanMode = TRUE;
}

void CTactiVisionDlg::OnContextMenu(CWnd* pWnd, CPoint point)
{
		GUISelector.TrackPopupMenu(TPM_LEFTALIGN | TPM_LEFTBUTTON | TPM_RIGHTBUTTON,
					(point.x), (point.y), this);
}

void CTactiVisionDlg::OnGetColor()
{
		CRect drawWindow;
		this->GetWindowRect(&drawWindow);
		hasColor = TRUE;
		CPoint topLeftClipPoint (zoomhex[0].getVertex1().x + 7 + drawWindow.left,
					zoomhex[0].getVertex2().y + 27 + drawWindow.top);
		CPoint bottomRightClipPoint (zoomhex[914].getVertex3().x - 2 + drawWindow.left,
					zoomhex[914].getVertex5().y + 16 + drawWindow.top);
		CRect clipRegion (topLeftClipPoint, bottomRightClipPoint);
		::SetCursorPos(((topLeftClipPoint.x + bottomRightClipPoint.x)/2), ((topLeftClipPoint.y +
					bottomRightClipPoint.y)/2));
		::ClipCursor(clipRegion);
		inGetColorMode = TRUE;
}
```

```
void CTactiVisionDlg::OnUpdateButton()
{
	char	colorBank[19];
	COLORREF colorREFBank[19];

	colorREFBank[0] =  zoomhex[lastFrame-60].getColor();
	colorREFBank[1] =  zoomhex[lastFrame-59].getColor();
	colorREFBank[2] =  zoomhex[lastFrame-58].getColor();
	colorREFBank[3] =  zoomhex[lastFrame-31].getColor();
	colorREFBank[4] =  zoomhex[lastFrame-30].getColor();
	colorREFBank[5] =  zoomhex[lastFrame-29].getColor();
	colorREFBank[6] =  zoomhex[lastFrame-28].getColor();
	colorREFBank[7] =  zoomhex[lastFrame- 2].getColor();
	colorREFBank[8] =  zoomhex[lastFrame- 1].getColor();
	colorREFBank[9] =  zoomhex[lastFrame].getColor();
	colorREFBank[10] = zoomhex[lastFrame+ 1].getColor();
	colorREFBank[11] = zoomhex[lastFrame+ 2].getColor();
	colorREFBank[12] = zoomhex[lastFrame+28].getColor();
	colorREFBank[13] = zoomhex[lastFrame+29].getColor();
	colorREFBank[14] = zoomhex[lastFrame+30].getColor();
	colorREFBank[15] = zoomhex[lastFrame+31].getColor();
	colorREFBank[16] = zoomhex[lastFrame+58].getColor();
	colorREFBank[17] = zoomhex[lastFrame+59].getColor();
	colorREFBank[18] = zoomhex[lastFrame+60].getColor();

	for (int index=0; index<19; index++)

	colorBank[index] = (char) (255 - (colorREFBank[index] & 0xFF));// + calibrate[index];

	// Serial Port Communication Setup
	comHandle = CreateFile(_T("Com1"), GENERIC_READ | GENERIC_WRITE, 0, NULL,
				OPEN_EXISTING, 0, NULL);
	SetupComm(comHandle, 1, 1);
	GetCommState(comHandle, &serialDCB);
	serialDCB.BaudRate		= 9600;
	serialDCB.ByteSize		= 8;
	serialDCB.Parity		= NOPARITY;
	serialDCB.StopBits		= ONESTOPBIT;
	serialDCB.fAbortOnError	= TRUE;
	serialDCB.fRtsControl	= RTS_CONTROL_HANDSHAKE;
	SetCommState(comHandle, &serialDCB);

	WriteFile(comHandle,colorBank,19,&bytesWritten,NULL);
	CloseHandle(comHandle);
}
```

```
class Hexagon
{
private:
        CPoint vertex[7];
        COLORREF hexcolor;
public:
        Hexagon();
        CPoint getVertex1() { return vertex[0]; }
        CPoint getVertex2() { return vertex[1]; }
        CPoint getVertex3() { return vertex[2]; }
        CPoint getVertex4() { return vertex[3]; }
        CPoint getVertex5() { return vertex[4]; }
        CPoint getVertex6() { return vertex[5]; }
        CPoint getCenter()  { return vertex[6]; }
        COLORREF getColor() { return hexcolor;  }
        void setHexagon(int xcoord, int ycoord, COLORREF color);
        void setColor(COLORREF color) { hexcolor = color; }
        void FillPoly(CDC* screen);
};

class CTactiVisionDlg : public CDialog
{ public: CTactiVisionDlg(CWnd* pParent = NULL);

        //{{AFX_DATA(CTactiVisionDlg)
        enum { IDD = IDD_TACTIVISION_DIALOG };
        //}}AFX_DATA

        //{{AFX_VIRTUAL(CTactiVisionDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        //}}AFX_VIRTUAL

        CMenu GUISelector;

protected:

        HICON   tvIcon;
        HACCEL  keyStop;

        HANDLE  comHandle;
        DCB     serialDCB;
        DWORDbytesWritten;

        BOOL inDrawArea;
        BOOL inPanMode;
        BOOL inGetColorMode;
        BOOL cursorHidden;
        BOOL hasPanned;
        BOOL hasColor;
```

```
        Hexagon zoomhex[915];
        Hexagon colorHex;

        COLORREF currColor;
        CRect lastPan;

        CDC  realThumbMemory;
        CDC  zoomThumbMemory;
        CDC  GUILayoutMemory;
        CDC* screen;

        int lastFrame;
        int lastColor;
        int topInBox;
        int leftInBox;

        BOOL PreTranslateMessage(MSG* incoming);
        void SetHexagons();
        void ColorPixels();
        void OutputPixels();
        void GetDrawing(CPoint point);
        void FrameHandle(int centerHex, CPen* currentPen);
        void PanDrawArea(CPoint point);
        int  GetHexagon(CPoint point);
        void GetColorHandle(int centerHex, CPen* currentPen);
        void GetColorDrawArea(CPoint point);

        // Generated message map functions
        //{{AFX_MSG(CTactiVisionDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnLoadButton();
        afx_msg void OnPanButton();
        afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
        afx_msg void OnSaveButton();
        afx_msg void OnGetColor();
        afx_msg void OnUpdateButton();
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

return to Section2.2 …

Appendix B

PIC16F628 Firmware Code

```
;*****************************************************************************************************
;  TactiVision Control Software Version1.3
;  (C) Vindica Systems Inc 2003
;
;  Written by Mike McFarland
;  Last Modified: 03/26/03
;
;  Revisions from Version1.0:
;           - implemented indexed addressing
;           - added continuous refresh functionality
;           - increased pixel capacity from 16 to 19
;
;  Revisions from Version 1.1:
;           - added sorting algorithm to receive/DAC routine
;
;  Revisions from Version 1.2
;           - removed sorting/swapping routines due to design change
;
;*****************************************************************************************************

list      p=16f628
#include <p16f628.inc>

__CONFIG _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _MCLRE_OFF & _LVP_OFF &
_INTRC_OSC_NOCLKOUT

pin01           EQU     0x20            ; memory address of first pin height
nextpin         EQU     0x33            ; memory address of next pin to receive

clock           EQU     0x35            ; clock variable definition
subclock EQU    0x36                    ; subclock variable definition

saveW           EQU     0x37            ; register for saving working register contents on interrupt
saveFSR         EQU     0x38            ; register for saving FSR register contents on interrupt
saveSTATUS      EQU     0x39            ; register for saving STATUS register contents on interrupt

                org     0x000
                clrf    PCLATH          ; clear upper byte of program counter
                goto    initialize      ; branch to initialization

                org     0x004           ; interrupt vector received (initiate service routine)
push            movwf   saveW           ; save working register contents in saveW
                movf    STATUS,W        ; move STATUS register contents into working register
                movwf   saveSTATUS      ; and save in saveSTATUS
```

```
receive         btfsc   PIR1,RCIF       ; if receive interrupt flag has been set
                call    rxHandler       ; call the receive interrupt service
pull            movf    saveSTATUS,W
                movwf   STATUS          ; reload pre-interrupt value into STATUS register
                movf    saveW,W         ; reload pre-interrupt value into working register
                retfie                  ; return from interrupt

initialize clrf STATUS
                clrf    INTCON          ; clear all STATUS and interrupt flags
                clrf    PIR1
                bsf     STATUS,RP0      ; select register bank1
                clrf    PIE1            ; clear interrupt enable flags
                bsf     PCON,OSCF       ; enable internally generated clock
                bcf     STATUS,RP0      ; select register bank0

                clrf    PORTA           ; initialize portA & portB to zeros
                clrf    PORTB
                movlw   0x07
                movwf   CMCON           ; disable comparator functions of portA

                bsf     STATUS,RP0      ; select register bank1
                clrf    TRISA           ; clear tristate (output portA)
                clrf    TRISB           ; clear tristate (output portB)
                bsf     TRISB,1         ; set receive pin on portB
                bcf     STATUS,RP0      ; select register bank0
                bcf     PORTB,0         ; clear the 'receive ready' line
                call    loadorder       ; load the decoder output sequence

                movlw   pin01           ; initialize pin heights in memory to zero
                movwf   FSR
zeroed          clrf    INDF
                incf    FSR
                movf    FSR,W
                xorwf   0x33
                btfss   STATUS,Z
                goto    zeroed
                goto    refresh

main            btfss   PORTA,5         ; stay here until the PC is ready to send something
                goto    main            ; when it is, skip this loop call and enter setupRX

setupRX         movlw   0x20            ; load the location of the first address to receive
                movwf   nextpin
                bsf     STATUS,RP0      ; setup up the SPI hardware for receive
                movlw   0x19
                movwf   SPBRG
                bsf     TXSTA,BRGH
                bcf     TXSTA,SYNC
                bsf     PIE1,RCIE       ; enable receive flag
                bcf     STATUS,RP0
```

```
                bsf     INTCON,GIE      ; enable global interrupts
                bsf     INTCON,PEIE     ; enable peripheral interrupts
                bsf     RCSTA,SPEN
                bcf     RCSTA,RX9
                bsf     RCSTA,CREN
                bsf     PORTB,5         ; tell PC that PIC is ready to begin receiving

getyet          btfsc   PORTA,5         ; should vector to interrupt handler
                goto    getyet          ; PC will clear send signal when complete

                bcf     PORTB,5
                bcf     RCSTA,CREN      ; clear and disable all interrupts
                bcf     RCSTA,SPEN
                bcf     INTCON,PEIE
                bcf     INTCON,GIE

refresh         movlw   0x32            ; sequentially address decoder to enable DACs in sequence
                movwf   FSR
cycle           bsf     PORTB,7
                bsf     PORTB,6
                bsf     PORTB,4
                bsf     PORTB,3
                bsf     PORTB,0

                btfss   INDF,7
                bcf     PORTB,7
                btfss   INDF,6
                bcf     PORTB,6
                btfss   INDF,5
                bcf     PORTB,4
                btfss   INDF,4
                bcf     PORTB,3
                btfss   INDF,3
                bcf     PORTB,0
                call    timer

                movlw   0xD0            ; move pin-height value for current DAC onto PortA
                movwf   PORTA
                movf    FSR,W
                movwf   saveFSR
                movlw   0x20
                addwf   FSR,F
                btfsc   INDF,4
                goto    demux2
                movf    INDF,W
                movwf   PORTA
                goto    complete

demux2          movlw   0xD0
                movwf   orderindex
```

```
                btfss   INDF,0
                bcf     orderindex,6
                btfss   INDF,1
                bcf     orderindex,7
                movf    orderindex,W
                movwf   PORTA
                btfss   INDF,2
                bcf     PORTB,2

complete        movf    saveFSR,W       ; repeat for next address, next DAC, next value
                movwf   FSR
                call    timer
                bsf     PORTB,2
                movlw   0xD0
                movwf   PORTA

                decf    FSR,F
                movlw   0x1F
                xorwf   FSR,W
                btfss   STATUS,Z
                goto    cycle           ; until all 19 pins have been updated
                goto    main            ; return to main to wait for new values from PC

rxHandler:
                movf    nextpin,W       ; load address of next pin into working register
                movwf   FSR             ; copy it into the indexed addressing register
                movf    RCREG,W         ; transfer the contents of RCREG to working register
                clrf    RCREG           ; clear the USART receiver register once read
                movwf   INDF
                incf    nextpin,F       ; increment pointer to next pin location
                return

loadorder:                              ; preload the decoder addressing sequence
                movlw   0x01
                movwf   nextpin
                movlw   0x40
                movwf   FSR
loadall         movf    nextpin,W
                movwf   INDF
                incf    FSR,F
                incf    nextpin,F
                movlw   0x53
                xorwf   FSR,W
                btfss   STATUS,Z
                goto    loadall
                return

timer:                                  ; use subtimer loop inside timer for ~ 1 second delay
                movlw   0x00
                movwf   clock
```

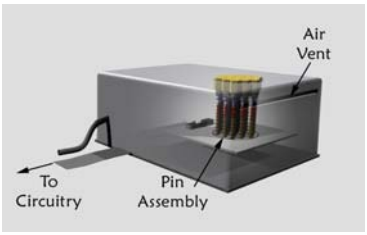
```
ping        incf    clock,F
            call    subtimer
            movlw   0xFF
            xorwf   clock,W
            btfss   STATUS,Z
            goto    ping
            return

subtimer:
            movlw   0x00
            movwf   subclock
pong        incf    subclock,F
            movlw   0x0A
            xorwf   subclock,W
            btfss   STATUS,Z
            goto    pong
            return

END
```

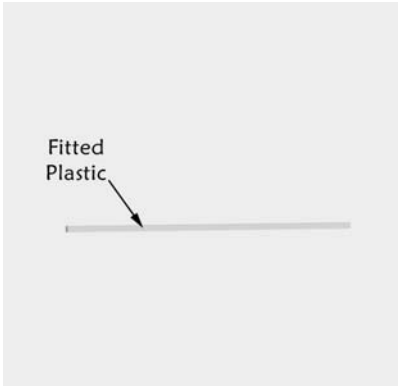
return to Section2.3 ...

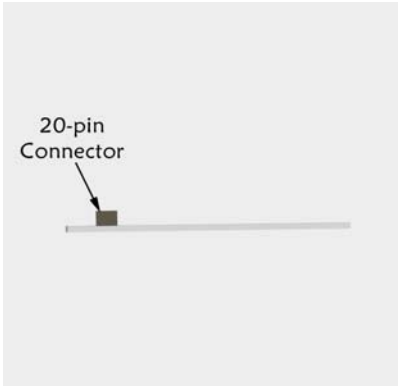
Appendix C

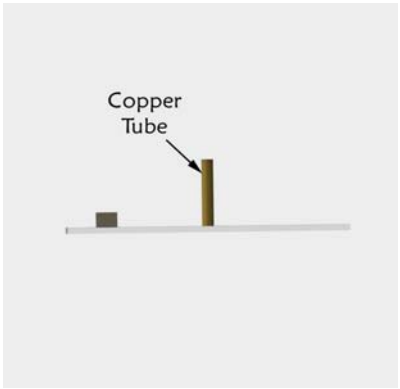
Tactile Display Construction

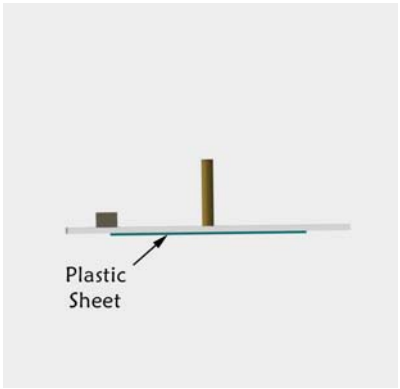
Pictures courtesy of Matt Lane © 2003

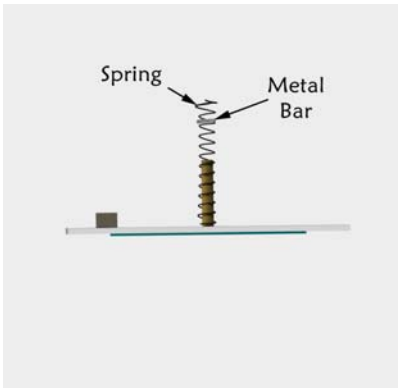


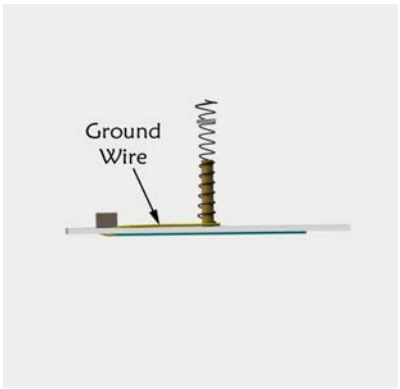


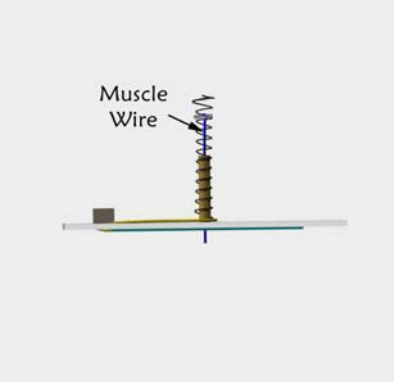


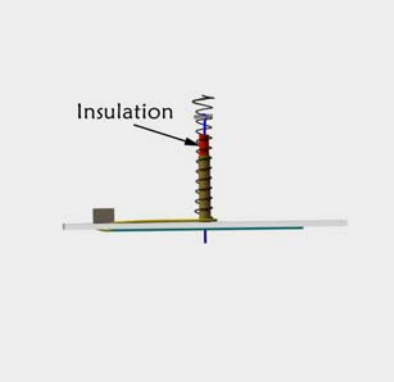


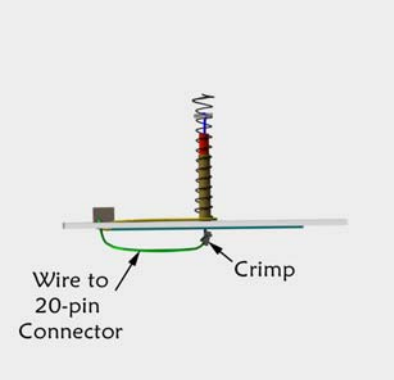


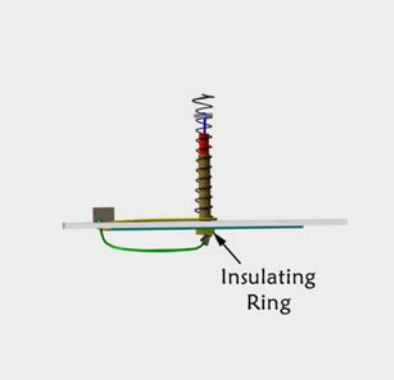


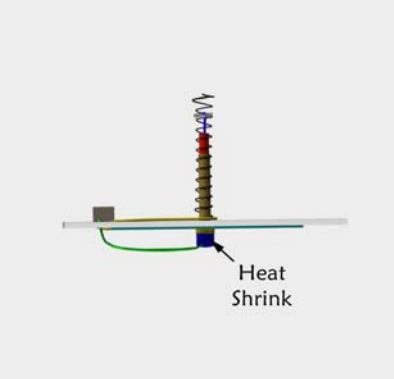


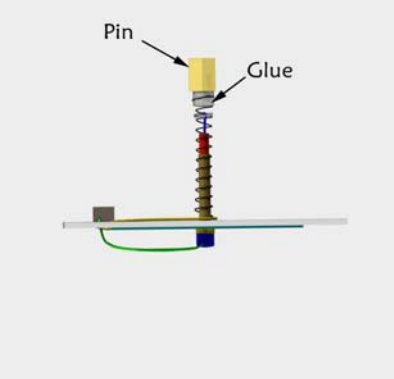


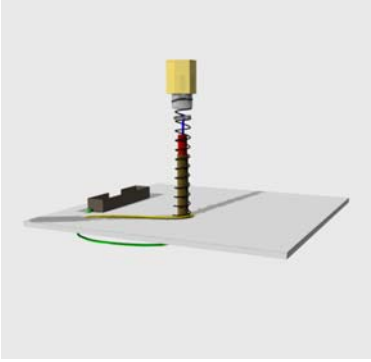








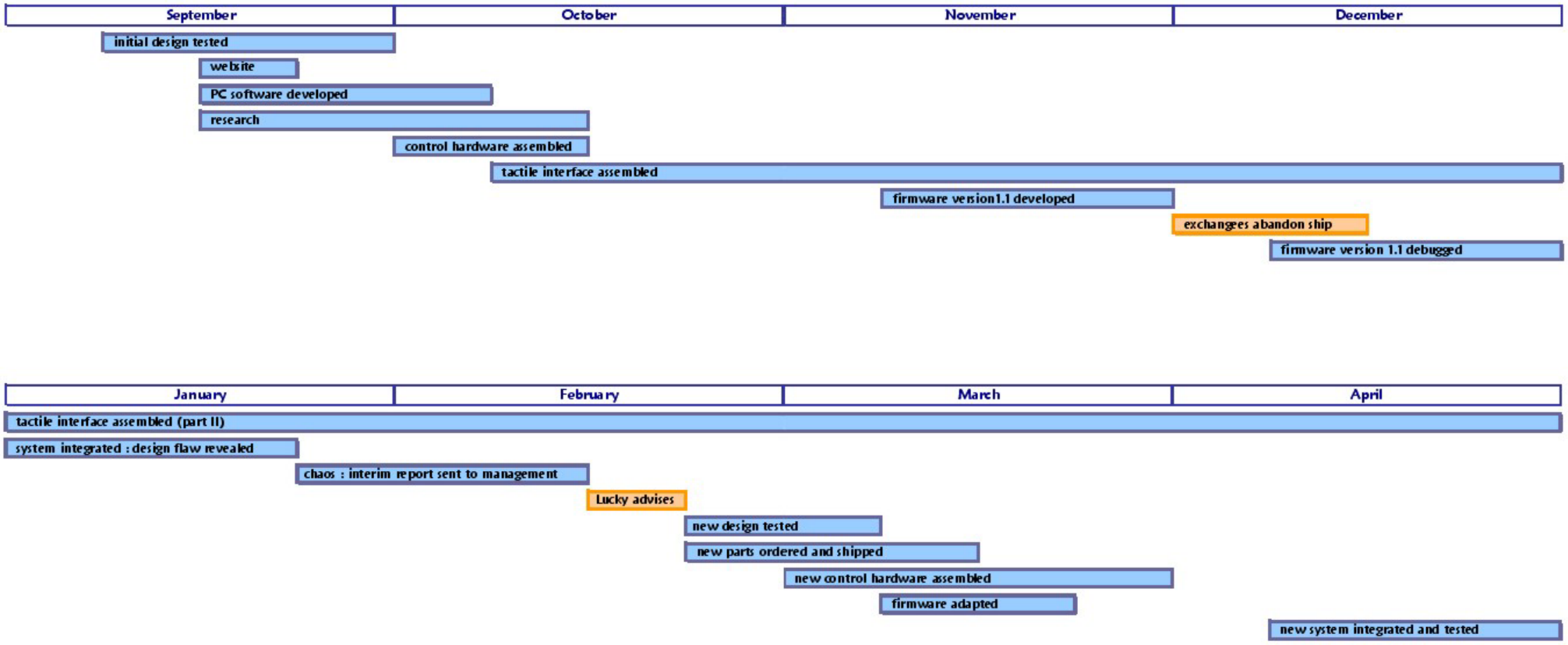




return to Section2.4 …

Appendix D

Project Timeline



return to Section4 ...