Design of a Communications Interface for E-Textile Buttons

Dave I. Lehn  
dlehn@vt.edu  
Craig W. Neely  
cneely@vt.edu  
Kevin Schoonover  
kschoono@vt.edu

29 April 2003

ECE 5984 - Wearable & Ubiquitous Computing  
Professor T. Martin  
2:00 TH - CRN 16445
# Table of Contents

Abstract 3
1: Introduction 3
2: User’s Guide for the Design 4
3: Project Design Decisions 5
4: Conclusion and Future Work 22
5: Appendix
   A. Section 1: Master PIC Code 23
   B. Section 2: Clicky PIC Code 28
   C. Section 3: Blinky PIC Code 31

*Notes:* A CD has been attached which contains all files related to this design project.
Abstract

This research presents the design and implementation of E-Textile "buttons". These buttons are small computing, sensor, or actuator devices integrated onto a PCB which is attached to an e-textile substrate. This research studies various attachment techniques and the physical realization of a communications strategy using the I²C bus.

1. Introduction

Electronic textiles, often called e-textiles, are textiles with integrated electronics devices, power resources, and data signaling. This research presents a study of a particular class of e-textiles. Computational, sensor, and actuator resources are integrated onto printed circuit boards (PCBs) and attached to a fabric e-textile for power and communication.

The goals for this study were:

- What type of connections should be used? Which is the least expensive? Which is the most reliable? Which is the most durable?
- What protocol should be used for communication?
- Implement a physical prototype which includes various button types communicating over a fabric textile.

This report includes three sections discussing the design process. The next section contains the user’s guide which provides an overview of the system along with a plausible demonstration scenario. The following section constitutes the bulk of the report and follows the design process through the various design decision steps in the project. Finally, the last section concludes the paper and highlights areas of potential future work and research.

This design is intended to provide three things; function-specific circuits implemented on a single button, a connection method to attach the buttons to an e-textile, and a means of establishing communications between the buttons. The idea behind the design was to create a system where various boards can be attached to an e-textile with relative ease also allowing some sort of communication between the buttons.

The system implemented currently uses three different devices (input, output, and communications master) to demonstrate communications over the textile. With all three devices connected, the input device can be used to send information to the master with the master then relaying this information to the output device. This system uses three buses connected to each other that provide ample space for several buttons to be connected. I²C is the communication method employed by the system. Different button attachment methods are used to demonstrate the strengths and weaknesses of each.

A typical demonstration would simply involve applying power to the circuit. Power application at one point on the bus will supply power to all devices on the bus. Once connected, each button can communicate with the master button. Pressing a button on the input device sends the switch status to the master which then relays it to the output device. The output button mimics the switch status with LEDs. Additional buttons can also be connected to the bus and incorporated into the system.
3. Project Design Decisions

Basic System Architecture

It was first decided to implement a design wherein all devices on the e-textile would have some degree of processing capability onboard. The integration of a microprocessor into the board designs allowed a greater degree of system flexibility to be achieved than would be possible with fixed analog circuitry.

Further narrowing the field, the use of microcontrollers was agreed upon as the primary processing family to investigate for use in the system, as they typically include the necessary I/O interfaces such as analog to digital converters and serial bus capability built in, on-chip memory, and an appropriate level of computational throughput for the low data rates required for envisioned applications. Other processing options such as DSPs would be good choices for more intensive operations such as acoustic beamforming or a mapper garment.

Our research into available microcontroller options led to several options including Microchip’s PIC, Atmel’s ATtiny, and other solutions such as a BASIC stamp from Parallax. The ATtiny and PIC microcontrollers were our two design finalists due to their small size, integrated analog to digital converters, FLASH-based program memory, and relatively fast instruction execution speed.

Two versions of the PIC microcontroller were finally chosen for our use in our system. The PIC16F819 was selected for its small 18-pin SOIC package and low power consumption for boards where computational power was not the primary design element. This microcontroller also offered an internal oscillator, reducing required external components. The PIC18F242 in a 28-pin SOIC package was selected for more intensive tasks such as a communications master device or where limited signal processing was required.

Connection Methods

One of the components of the design involved selecting a method of attaching the buttons to the e-textile. The following factors were evaluated when considering different connection methods:

- Physical strength
- Electrical reliability
- Ease of attachment
- Aesthetics
- Size
- Comfort
- Cost
- Availability
The different methods considered during the design were buttons or snaps, bolts, solder, and ribbon cable connectors. The above factors were considered when selecting a method to implement. The following sections discuss the pros and cons of each method.

**Solder**

Soldering the buttons to the e-textile was an option, but not a serious one. An extended goal of the design was to provide a way to attach the button to the textile that could be used in a mass production setting. The process of soldering each component or pin to the fabric would probably be too slow for mass production. A solder connection would most likely require more time if done by a person or would be overly expensive if performed by a machine. Soldering presented several other problems.

While the practice would produce a reliable electrical connection to the fabric, the physical strength of the solder connection would be questionable. The actual solder joint itself tends to stay in tact but the hardened solder also provides a bending point where the wire itself can break. Soldering may not even be an option for some types of threads. Some potential wires would not be able to withstand the heat of the soldering process.

Problems would also arise when attempting to align the pins on the buttons with the wires on the fabric. Connecting the pins of the board to the fabric is a slower process when soldering because each individual pin needs to be aligned with its corresponding wire. The connection process is not one single step; each wire has to be attached separately. With the dynamic shape of an e-textile to consider, the desired wires may not always be spaced evenly.

If the wires are insulated, the task of removing the insulation at the attachment point would also add to the complexity of the process. Additionally, to avoid exposure of contacts, the solder joints would then need to be reinsulated. Again, this adds time to the procedure as well as cost.

The benefits of soldering include limited size, weight, and effects on comfort and aesthetics. The connection size itself is reduced to the size of the additional solder. The reduced size minimizes the impact on appearance and comfort. Re-insulating the wires though, may have adverse effects on the appearance of the garment. Although soldering would provide the least bulky and most comfortable connection, the difficulties associated with it would outweigh its benefits.

**Snaps**

The snap method uses sew-on snaps to make the electrical connections between the board and the fabric. One side of the snap would be connected to the fabric wire, and the other side would be connected to the button. The snaps considered are commonly used in textiles today. Some of the benefits of this method include its already popular use. What is not common, however, is the connection of the snap to a wire woven into the fabric.

One possible method to connect the snap to the wire is with solder. The wire is either already exposed or the insulation has to be stripped from around the attachment point. This method would provide an adequate communication connection but the connection may become physically unstable. Sewing the snap to the fabric with thread would help stabilize the connection while also holding the snap in place.
on the fabric. Soldering, again, may not be an option due to the properties of some of the potential wires.

Welding the buttons to the wire is a technique similar to soldering. As with soldering wire compatibility with the welding process is also an issue. One of the advantages of welding over soldering would be, given the proper machine, the snaps could be connected to the wire in a more efficient manner than with using solder. Soldering requires the elements to be heated which slows the process. Welding provides a better chance for mass production to be feasible.

Another method of attaching the snap to the fabric wires is to use a wire thread. The wire thread could be used to sew the snap onto an exposed piece of the wire. With the possibility of the wire thread coming loose, the stability of this connection is uncertain. Theoretically, a machine could be used to puncture the insulation to allow the wire thread to come in contact with the conductor of the wire. This could be a bit more expensive. Questions are then raised as to whether sufficient contact area between the thread and the wire can be achieved.

Standard thread could also be used to hold the button against the exposed wire. This technique is even more insecure due to the lack of a conductor joining the two. The thread method relies on the tension in the thread to hold the snap against the wire. When the thread becomes loose, the connection becomes questionable at best.

Aside from their common use, an advantage of snaps is their ease of attachment and removal. If buttons are desired to be removed and added somewhat frequently, using snaps would be much easier than bolts, soldering, or welding. Repeated connect/disconnect cycling could weaken the strength of the connection, compromising the stability of the connection.

The large surface area of the snaps provides a decent platform for an electrical connection. Figure 3-1 shows the size of the snap connection pads. The downside of the larger connection is that it increases the size of the board. Since snap is needed for each wire connection, depending on the number of wires needed, the use of snaps could force the connection interface to be the dominating factor in the size of the board. Since size is an important constraint, the number of snaps needed needs to be considered before selecting this approach.

One problem common with any of these exposed-wire methods is the potential for wires to short. Even with insulated wires, the use of snaps provides an ever-present opportunity for two leads to touch. Insulating portions of the snaps may help but that only decreases the conductivity of the connection. If the snap connection is always being used by a button, however, the possibility of crossing wires at the snaps is almost eliminated.

Snaps seem like a reasonable solution given certain designs. Naturally, designs requiring more wires should probably stay away from the snap method. Snaps would also benefit systems where buttons
need to be switched in and out often but also where a snap connection usually will always have a button on it to avoid wire shorts.

**Ribbon Cable**

The ribbon cable connectors considered for this project utilize a connection method called insulation displacement. The connections are made within the housing of the connector through the use of a sharp V-shaped contact that cuts through the insulation to connect to the conductor. Figure 3-2 shows the insulation displacement method.

The use of insulation displacement removes the need for mid-wire insulation stripping. The plastic housing of the connector also provides insulation of the contacts. The sockets of this connector allow for easy mating with the pins of the buttons, providing a reliable connection with the convenience of easy removal. When the button is removed the contacts to the fabric are not exposed.

Successive pins in the connector could also be used for a common pin on the fabric. This would help reduce alignment troubles since more than one wire slot could be connected to a pin on the board. Using more connector pins would naturally increase the size of the connector, but if the connector size remains within the limits of the board, additional pins should not affect the design size. Connectors of this type with various numbers of pins are available so finding a connector to meet a specific need should not be difficult.

The ribbon cable connector itself can be removed with little difficulty. This, however, can lead to exposed portions of the wire or the insulation displacement could damage the conductor enough that the wire is in danger of breaking. A significant problem with these connectors is the potential of cutting the wire. If the connector remains intact, wire breaks should not be a problem. Though the connector can be removed, it would probably not be an intelligent idea in an actual system.

Generally, connecting this type of connector to a group of wires does not take that much physical effort. However, in the case where the insulation displacers need to cut through both the wires and the rest of the fabric, the force needed to fasten the connector together requires some sort of clamping tool.

Another disadvantage of the ribbon cable connector is its vertical size. Ideally, a connector should be customized to reduce its vertical dimension. The plastic housing of the connector considered could easily have been reduced. One idea would have the insulation displacers bent at a right angle so that one side can be soldered to the board and the other side can connect to the wire. This would eliminate the use of the plastic housing but would also remove some of the contact exposure protection. Having some sort of structure under the insulation displacers would help align the wires and ensure the connection.
Bolts

The bolt method involved using a bolt and nut to clamp a copper tube down on the wire in the fabric. Both the bolt head and the copper tube would make contact with the board on one end. The tube and a washer completed the connection at the wire. The bolt method is shown in Figure 3-3.

Some of the benefits of the bolt method included a stable physical and electrical connection. If the copper tube could incorporate some sort of insulation displacement mechanism, attachment of button would be as simple as threading the bolt through the tube, aligning the slot in the tube with the wire, and tightening the nut. If the insulation is not displaced by the tube, then the undesirable task of mid-line insulation removal would be necessary. Attaching and removing the button would of course require tools since a hand-tight connection would not be sufficient in most cases.

As is common with nuts on bolts, the connection may lose stability over time as the nut begins to gradually turn off the bolt. This problem could be remedied with the use of locking nuts, but then the disconnection possibilities of the button would be compromised.

Other drawbacks to the bolt method include its size, weight, and discomfort characteristics. Although the tube, bolt, nut, and washer could be sized down somewhat, the setup would still be somewhat bulky and heavy compared to other methods. There would also remain a problem with the inside ends of the bolts being uncomfortable against the user’s body.

Table 3-1 below summarizes the advantages and disadvantages of each method. The ribbon cable method was selected for its reliability and ease of implementation.

<table>
<thead>
<tr>
<th>Type</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solder</td>
<td>solid electrical connection, strong physical connection, small, light, comfortable, not noticeable</td>
<td>slow connection process, wire compatibility issues, wire breaks, alignment issues, mid-wire stripping, exposed wire protection, expensive</td>
</tr>
<tr>
<td>Snaps</td>
<td>connection/reconnection ease, common use</td>
<td>slow connection process, soldering or welding issues, connection size, weak physical connection, exposed leads</td>
</tr>
<tr>
<td>Ribbon Cable</td>
<td>insulation displacement, common part, insulated connection, alignment tolerance, reliable connection</td>
<td>size, installation difficulty, wire breaks</td>
</tr>
<tr>
<td>Bolts</td>
<td>solid electrical and physical connection, possible insulation displacement</td>
<td>removal difficulty, size, weight, discomfort, long-term connection stability</td>
</tr>
</tbody>
</table>
The Sweater

In order to test the connections, buttons, and communication a generic e-textile substrate was created. The garment used for this project was a small sweater with three buses woven into it. The project did not require optimal textile fabrication so a simple model was created. To line up with the connections on the buttons, four 28 AWG wires spaced 0.15 inches apart were used for each bus. The wires were woven into the sweater fabric in an attempt to mimic a typical e-textile.

Figure 3-4 below shows the layout of the test e-textile. Figure 3-5 shows the final implementation on the e-textile sweater. Two vertical 4-wire buses were run on either side of the front of the garment. One horizontal 4-wire bus extended across the chest and down both sleeves. The master, Blinky, and Clicky boards were arranged on different parts of the system to test the connectivity of the bus links. Basically, the master board handles the I2C bus communication, Mic samples audio data and puts it on the network, Blinky is an LED-based output board, and Clicky is a switch-based input device. A more detailed explanation of the different boards will be given later in the paper.

![Figure 3-4: Test E-Textile Layout (M = Master, B = Blinky, C = Clicky, Mic = Microphone)](image)

![Figure 3-5: Final Sweater System](image)

The connection where Blinky is located uses the snap method. All other button connections use the ribbon cable connector. All but one of the ribbon cable connectors were attached with no difficulties. A crimp tool was used to provide the necessary force to cut through the insulation and sweater thread. The redundant pins used by the connector made up for some imperfections in the bus line spacing. One problem encountered when attaching a ribbon cable connector was with getting the first pin to make contact with wire 1 of the bus. It was first though that clamping the connector on one side first...
was causing the insulation displacers to misalign with the wire. It was then discovered that one of the solder bus connections had gone bad. Overall, though somewhat difficult to squeeze together, the ribbon cable connector proved to be a satisfactory method.

The snap connectors used for Blinky were a bit more difficult to attach. First, insulation had to be stripped from around the connection point. The snap was soldered to the wire and then sewed to the fabric. Each of the four snaps had to be attached separately. Though sewing and soldering inexperience probably contributed to the attachment time, it still would have taken considerably longer than the ribbon cable connectors.

The other problem with connecting the snaps to the bus was aligning the snaps to match the pattern found on the snap board. Figure 3-6 shows the snaps sewn onto the sweater along with the snap board used to connect to the snaps. Though the fabric is still flexible, the snaps still had to be spaced similarly to the snaps on the connection board. Spacing too far apart may require stretching of the garment which could break connections. Spacing too close may make it impossible to connect to the board since the board cannot be collapsed to meet the fabric connection.

Overall, the use of the ribbon cable connectors on the fabric was satisfactory. A more customized connector could have been used and probably should be used in the future, but for the sake of prototyping the connectors used were sufficient. The expected faults in the other methods were observed in the experimentation process. From the techniques considered, some form of an insulation displacement connector seems to be the better choice for connection.

**Bus Connections and Cross-Seam Connections**

Another design consideration was the necessity of connecting crossing buses or making connections across seams in a textile. Buses in the textile will not necessarily all run in parallel, nor will separate buses necessarily be able to remain separate. Buses in separate regions of a garment, the sleeve of a shirt perhaps, may need to be connected to another region across a seam in the fabric. Connecting these buses allows the components to communicate with each other and allows one common bus to be used throughout the textile.

Two methods of bus connection were considered for this project. The first method, implemented at the junction of the horizontal bus and one vertical bus, used soldering to connect the wires of the two buses. One attempt at connection involved stripping a section of insulation out of the middle of the wires and soldering the exposed portions of wire together. This proved to be quite difficult as the process of removing a center portion of insulation was not fast or simple. The other method was to actually cut the wires and solder the four pieces together. This removed the insulation stripping dilemma but also made the connection slightly weaker.
Although the joints could be hidden reasonably, the exposed connections had to be taped to avoid crossing wires. Figure shows the soldered bus from the inside of the sweater. Even though the soldering was done on the inside of the sweater, the added tape made the connections a bit unsightly when viewed from the inside. The tape also enlarged the connection size and forced some stretching and pulling of the fabric threads, distorting the appearance of the garment somewhat.

Another problem with the solder joints was their lack of reliability. Connections had to be repaired twice as the result of pulling and stretching of the fabric. The instability of the connection caused some uneasiness in handling the fabric, which should not be the case. One of the purposes of having a textile-based system is to take advantage of its collapsibility. If the connections within the textile are too fragile to allow bending or pulling, the deformability feature of the e-textile cannot be utilized.

A final argument against the use of solder to connect the buses was its lack of mass production potential. Neither the mid-wire stripping nor the wire cutting method would be easily or cheaply implemented in an automated process. This process would introduce numerous alignment problems. As wires in the bus get closer together, assuring that all the correct connections were made would be difficult.

The other approach to joining two buses used two of the ribbon cable connectors, one on the horizontal bus and one on the vertical bus. Figure 3-8 shows this connection method. The appropriate connections between the two ribbon cable connectors were then made using single wires. This allowed for some flexibility in the joint while providing a reasonably stable electrical connection. Ideally, a mating ribbon cable could be used to join the two connectors. This would enhance the stability of the connection as well as improve its appearance. No problems were encountered when making this bus connection.

This same approach could be used to connect buses across seams. A bus could lead up to one edge of the seam and could be terminated by one of the ribbon cable connectors. A similar ribbon cable connector would then be attached to the end of the bus on the other side of the seam. A flexible ribbon cable would then allow a suitable connection across the seam. The ribbon cable would allow the seam to flex while still providing a reliable connection.

**Communications Decisions**

With the physical connection types between devices selected, the next step in the project design process was to select the communications protocol that would be used to interconnect the devices. The two main areas of consideration were the communications standard to be used and the hierarchical structure of the devices in the network. As the direct transmission of analog data signals over a potentially lossy medium as might be found in the wiring on e-textiles would not be desirable due to losses in signal resolution, only digital transmission types were considered.

Several different possible communications standards that were explored for use in the project are summarized in Table 3-2. The first was the simple parallel connection of data between devices, which
would allow for dedicated lines to facilitate the transfer of data, removing requirements for device
handshaking and time division access on the bus. In some situations this could form the basis for a fast,
stable link, however in an e-textile environment where many devices need to be networked together
and the manufacturing complexity goes up with the increased wire counts necessary for parallel
connections, a serial bus interface was selected for use.

Several different serial connections were considered including the inter-integrated circuit (I\(^2\)C) bus, the
serial-peripheral interface (SPI), a controller-area network (CAN) bus, or familiar computer serial links
such as USB and Firewire. Our requirements for the link were that it be easy to implement, capable of
supporting high enough data rates to network many simple e-textile buttons, be implemented with a
low wire count, and be reasonably fault tolerant.

The SPI, I\(^2\)C were considered finalists for the selection process as they were all easy to implement with
our selected processors as Microchip’s line provides devices with native support for all three
interfaces. USB, Firewire, and CAN bus offer the highest maximum data rates and most robust
connections of the serial links, but were removed from contention because of the complexity and
special handling required to implement these protocols.

An SPI interface is marked by separate input and output serial data lines and an additional device
select line for each device in the network. This type of interface has the benefit of reducing protocol
complexity, but suffers because of the high number of wires required for a large network.

<table>
<thead>
<tr>
<th>Type</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel</td>
<td>* Increased bandwidth</td>
<td>* Specialized connection * Increasing wire count and complexity with additional devices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* EM interference</td>
</tr>
<tr>
<td>I(^2)C</td>
<td>* Supported by PIC µCs * Previous design knowledge * 4-wire interface (+,-,SCL,SDA)</td>
<td>* Designed for board level integration, limited handling of longer distances</td>
</tr>
<tr>
<td>SPI</td>
<td>* Supported by PIC µCs * Independent data I/O improves simplicity, speed</td>
<td>* Requires additional select line for each extra device</td>
</tr>
<tr>
<td>CAN Bus / USB / Firewire</td>
<td>* More suited to off-board comms. * Faster, more robust</td>
<td>* Additional complexity * Extra time to develop</td>
</tr>
</tbody>
</table>

Finally, I\(^2\)C was selected for the system’s communications interface. I\(^2\)C is a 4-wire interface,
consisting of shared power, ground, clock, and data connections. It is fairly fault tolerant as devices
always check the bus to make sure it is idle before commencing data transmission. Error checking
support is also provided by the PIC I\(^2\)C implementation. The link’s maximum speed in set as
implemented at 400 kbps, which is certainly high enough for simple data transmission such as the user
input device, Clicky, designed in the test system, but provides only enough bandwidth to handle a few beamforming microphones. This is a limitation that will need to be addressed with the use of higher speed busses as the project becomes more refined.

Software Development

Master PIC Code

The Master board serves as the controller for the entire serial network. It polls devices as necessary and either transmits data to or receives data from the slave boards. In its initial implementation, the master queries the Clicky data input board for the current switch status and then transmits this data to the Blinky display board. In the future this board will serve to route data from multiple boards, including the microphone board currently under construction. It will also interface via the onboard universal asynchronous receiver transmitter (UART) module to the Serial board and to a user’s PC for external control and monitoring of the fabric network system.

At present, the master code is organized into several common I²C functions for both reading from and writing data to the slave devices. Control functions include i2cStart, which pulses the data line from high to low while the clock line is left high, i2cStop, which pulses the data line from low to high while the clock line is left high, checkIdle, which probes for a high and unused data line before taking control of the bus, and initiateAck / initiateNack, which send the master’s reply to called slave devices marking that the master has either received data or is finished with a given data transfer set.

Data writing functions include i2cWriteAddr, which transmits the address of the desired slave device on the data bus and i2cWriteData, which writes one byte of data to the addressed slave. Finally, i2cReadAddr addresses the desired slave device to request a read sequence, i2cRead reads a single data byte from the buffer and acknowledges the transmission from the slave by calling initiateAck, and i2cReadLast reads the final data byte from the link and calls initiateNack to signal that transmission is complete.

In its present implementation, the Master board also mirrors the current switch status onboard the Clicky device on its own LEDs.

Clicky PIC Code

Setting up code for an I²C handler for the Clicky data input board was fairly simple by comparison to the Master. As Clicky has no foreknowledge of the time at which an I²C interrupt may take place, an interrupt service routine was developed to immediately process incoming communications requests from the master.

On an interrupt the I²C handler checks the serial status register of the PIC to make sure that a read request has been submitted from the master, the buffer is currently full with the matching address of the Clicky board, and that no write collision errors have occurred. A single byte indicating the status of the two pushbuttons switches will then be sent out over the link by moving the desired value to the serial buffer, where it is shifted out in serial format.
In its present implementation, the Clicky board also mirrors its own switch status on its own LEDs, and uses its one red LED as an error lamp on write collisions, which serve to indicate timing errors or link failures.

**Blinky PIC Code**

The interrupt handler for the Blinky display board required a new state machine implementation given the two-byte data format it receives from the master over the serial link. One state notes the reception of a matched address and a request for the slave to receive data from the master, while the other indicates that valid data is ready in the serial buffer. The interrupt handler services these two conditions by reading any data in the buffer and releasing the clock line for the master’s use. This technique is called clock stretching, and is used in more complicated code sections to ensure that any computations made by the slave device can complete before the master proceeds.

After the Blinky board receives a byte indicating the current switch status, it combines the two pertinent lower bits of the transmitted byte with other bits controlling the remaining LEDs on the board and writes this data to an output port. This has the effect of again mirroring the switch status on two LEDs.

**Button Design**

For this research a few types of buttons were designed and built. Examples from various classes of device were chosen for implementation. There are many types of sensors and actuators that can be used but a few types are sufficient to demonstrate functionality.

**Buttons**

- **Master**: Many algorithms require centralized computational resources. The Master button includes a larger microcontroller for such purposes. It also has the ability to do serial port communication to an external source when it's built-in UART port is connected to the Serial Transceiver button.

  ![Figure 3-9: Finished Master Button Front and Back Sides](image)
• **Blinky**: This button demonstrates basic output to the user by means of an array of lights.

*Figure 3-10: Finished Blinky Button Front and Back Sides*

• **Clicky**: Basic user input is accomplished through the use of a few switches on a button. User input devices could be more complex and include devices such as accelerometers.

*Figure 3-11: Finished Clicky Button Front and Back Sides*

• **Serial Transceiver**: This button is a simple small RS-232 transceiver. To reduce cost this button does not have computational resources. It is expected to have it connected to a UART enabled button as needed. One such device in this research is the Master button.

*Figure 3-12: Finished Serial Transceiver Button Front and Back Sides*
• **Mic**: Mic is an example of a passive high bandwidth analog environmental sensor. It uses a microphone, bandpass filter, and analog amplification circuitry to monitor audio.

*Figure 3-13: Finished Microphone Button Front and Back Sides*

An early decision was made that the fabric prototype would use \( I^2C \) as the communications network. This decision reduced the choice of processors to those that included native \( I^2C \) support. All but one of the buttons includes a microcontroller with built in \( I^2C \) support. Microchip PIC16LF819 microcontrollers are used for the "slave" devices while a Microchip PIC18LF242 is used for a larger "master" device. The master includes hardware support for \( I^2C \) master mode. The slaves can implement master mode in software.

These main boards are designed to be used in a prototype system. The data, power, and programming signals are all done through a standardized 0.100 inch pin header. This allows the true connection to the e-textile substrate to be changed easily during experimentation without significant modification to the more complex button boards. A few additional boards were manufactured for this purpose along with a programming support board.

• **Pin Adaptor**: Used to connect to ribbon cable type connections on the fabric.

*Figure 3-14: Pin Adaptor Button Front and Back Sides*

• **Bolt Adaptor**: Through holes and copper tubing are used to clamp down on wires.
• **Snap Adaptor**: Simple conductive fasteners which can be sewn on.

• **Programming Adaptor**: To convert the Microchip ICD2 debugger/programmer modular jack connection into a more compact pin header format.

**Power Regulation**

One important design issue is power regulation. There are two possibilities. First is global power regulation. This involves a centralized power source and regulator. The regulated power is sent out on
the e-textile to individual nodes. At first look this does reduce complexity of the power consumers. Only one regulator is needed. However, the major drawback of this method is that a node can be affected by remote power fluctuations and noise sources.

A second method is local power regulation. This adds hardware to individual nodes but has a number of advantages. The first of these is that nodes can operate at whatever voltage they need. Signal lines still must follow some standard. The input power lines to each node can also be less controlled. In a large e-textile this may be unavoidable due to environmental factors. Local filtering of power may be required even with global regulation. Some power regulators are available in very small packages to minimize board space.

For this research local power regulation is used. Each board includes a Microchip TC55 power regulator and filtering capacitors. On the current buttons the power regulation circuitry occupies a small but noticeable amount of space. With debugging components removed and smaller parts it may become quite significant. A possible solution is to have a shared local power regulation button for a small number of local buttons. This would have the drawback of a common failure point but may be acceptable in some applications.

**Debugging**

For debugging purposes with initial prototypes all the buttons include lights. This is very helpful to check basic functionality. It does, however, increase size, complexity, and power usage of simple buttons.

**Button Fabrication**

A major goal of this research was to build fully working prototypes. This was achieved by going through a full design and fabrication process. The fabrication of the boards is not directly related to the button theory but took significant time and effort. A brief description of the process follows.

For this project in-house PCB fabrication equipment was used. Once the schematics and board layouts are finished the next step is to convert the layouts into a format which can be used by the fabrication equipment. One tricky issue with the button designs is their small size. Making single boards is not practical due to overhead of the whole process. Therefore the boards are tiled and combined into a larger layout. The standard for almost all PCB equipment is the Gerber format for various layers and Excellon for drill information. Combining individual drill and layer information from many different small boards is challenging, if not impossible, in most software tools. Andrew Sterian has written a tool called GerbMerge[ http://claymore.engineer.gvsu.edu/~steriana/Python/] which does panelizing specifically to put smaller boards together for manufacturing. This tool is ideal for combining many button boards for fabrication.

To utilize available material it was determined that a roughly 2.1 inch by 9 inch section of copper plated PCB material was available for boards. The nature of the small button boards is ideal for making use of otherwise scrap material from other prototype boards. It was decided that at least one of each board would be manufactured. The fabric connection boards were chosen to provide a demonstration
of various attachment techniques but assume that the ribbon cable and pin header style would be easiest to assemble. The tally of manufactured boards includes the following:

- 2 x Masters
- 2 x Clicky
- 2 x Blinky
- 2 x Serial Transceiver
- 1 x Mic
- 4 x Pin Adaptor
- 1 x Bolt Adaptor
- 1 x Snap Adaptor
- 1 x Programming Adaptor

Once the board information is merged the next step is to drill the vias, through holes, and other holes in a copper plated PCB. Registration holes are first drilled so that the drilling and milling process are aligned. Then the board must be electroplated to provide conductivity through the holes. This step is done by first filling the holes with a conductive silver-based ink. Then the ink is blown through the holes so that it only remains on the edges of the holes. Next the board is placed in an oven to dry the ink. Then the board is placed in an electroplating tank where current is run through the copper on both sides of the board. The conductive ink in the holes is then plated with a layer of copper. At this point the board is ready for milling. This process involves adjusting milling tool cutting depth such that the fine traces are properly cut out. Finally, a router bit is used to cut out the boards from the surrounding material.

Creating the button PCBs was very tedious and error prone due to the nature of the entire PCB fabrication process. The first attempt at electroplating failed a visual inspection of about 90% of the holes. They appeared dark as if no copper plating was present. A decision was made to re-plate the board. After the second plating it appeared that less than 10% of the holes were plated but likely that almost all had some electrical conductivity. This is basically impossible to check before milling out traces.

The milling process itself is rather difficult. The first step for milling machines is to create isolation paths. This specifies the areas to remove around the copper traces. In order to make soldering easier the buttons use a two pass system. The first creates a 0.010 inch isolation. The second is setup to do a 0.025 inch isolation but is performed with the same 0.010 inch tool which creates a total of 0.0175 isolation. In the board layout phase the design rules were setup to allow 0.008 inch features. However, the milling tool software incorrectly identified the smallest feature as greater than 0.010 inch. This resulted in the need to later return and use a very fine milling tool to cut the handful of 0.008 inch traces on the boards.

Extreme care must be taken to properly adjust the milling tool depth. Too high and the electroplating and copper layers are not cut through. Too low and a wide path can nearly or fully cut away a trace. To make matters worse, slight height variances across the entire board can require that milling be done in stages such that right to left or front to back tool depth adjustments can be made. The safest approach to this problem is to err on the side of shallow cuts. This avoids problems with eliminating traces completely. However, it can, and did, result in the need for multiple passes over parts of the board.
Tool wear and tear is also an issue. The tools used to cut fine traces of 0.010 inches or less do not last very long. Fabricating the button PCBs nearly used up 2 milling bits. At $15 a piece this must be taken into consideration. Ideally this can be somewhat optimized with the use of end mill tools and other techniques to reduce wear on the fine tools. Various milling machine issues caused this to not be an option for this project.

After all the milling was complete the result boards turned out very good. Only one via was unconnected. This was easily fixed by soldering a rework wire through the hole. A few spots of shallow milling were cleared up with a sharp knife and microscope. Two fine traces were cut completely though. They looked as if they were milled through on purpose but were clearly errors. The cause of this is unknown. A number of traces were nearly cut through. One was thin enough to have measurable resistance with a basic multi-meter. These traces were cleaned up by adding a small amount of solder using a high quality soldering station and microscope.

The end result is a handful of button boards that work. Making the boards in-house provided good experience in this type of work. The process is acceptable for a small number of prototypes but making boards in quantity should be done commercially. Commercial production would offer, at a minimum, solder masks, more accurate fine features, and quality control at less expense and likely less time.
4. Conclusion and Future Work

The project goals of creating an efficient, stable, and highly customizable communications link between e-textile buttons was completed successfully.

The ribbon cable connectors selected as the physical connection between the e-textile serial bus wires and the buttons proved to be a fairly robust solution to the problem of integrating “soft” fabric good to traditional electronic devices mounted on printed circuit boards. These connectors provided three redundant circuits per wire in the bus, decreasing the required manufacturing alignment tolerances, and offered a cheap and readily available solution that could be attached to the underlying wires in an e-textile simply by pressing the insulation displacement knives into the fabric.

A more elegant solution to the physical connection problem may lay in the development of a customized variant of the ribbon cable connector as in its current form, the physical height of the connector may be considered too bulky for use by some users. Its height is primarily due to the length of the pin header that is inserted into it, but this may be bypassed by producing a version that simply consists of the insulation displacement scissors, and a direct connection to the circuit board. A removable solution consisting of rivet sized snaps is also a possibility for “plug and play” type operation.

The I²C serial communications link chosen for the project is an excellent bus type for networks requiring less than 400 kbps total bit rate for all devices. It is easily implemented and requires only 4 wires throughout the system. Future implementations will clearly need increased available bandwidth for signal processing applications and greater numbers of devices connected to the network, warranting study into high speed links such as USB and Firewire. Additionally, advanced multi-master network architectures would allow individual devices to broadcast messages over the bus which would be received only by interested devices, freeing the system from the master-slave hierarchy currently in place.

If the design process were to be done over, a major decision change would be to have the boards fabricated professionally. The fabrication process took a large portion of the total design time, reducing the amount of time available for increasing functionality. Different communication methods could have been explored in more detail. The microphone and serial interface may also have been implemented by the end of the design time.

The system created is highly scalable and is well suited to many different e-textile applications. It is relatively cheap to implement, and consists of devices that are standard to the industry, providing easy integration into fabrics. Our project design group hopes that this system will be used as a base platform for future development of digital serial communications links for e-textiles.
Appendix A: Master PIC Code

;***********************************************************************************
; MASTER.ASM
; Revision: B, 4/28/2003
; Status: complete
; Future: add microphone & UART support
;***********************************************************************************

; Port Usage
;***********************************************************************************

; RA: [6] - N/C
; [5] - N/C
; [4] - N/C
; [3] - N/C
; [2] - N/C
; [1] - N/C
; [0] - N/C
; RB: [7] - programming data
; [6] - programming clock
; [5] - N/C
; [4] - LERR
; [3] - LED3
; [2] - LED2
; [1] - LED1
; [0] - LED0
; RC: [7] - serial RX
; [6] - serial TX
; [5] - N/C
; [4] - I2C data
; [3] - I2C clock
; [2] - N/C
; [1] - N/C
; [0] - N/C

;***********************************************************************************
; I2C Interface Notes - Master is an I2C Master
;***********************************************************************************

; I2C master will read clicky buttons for switch values
; I2C master will send switch values to blinky
; FUTURE: READ FROM MIC

;***********************************************************************************
; Include Files
;***********************************************************************************

list p = 18f242
include <p18f242.inc>

;***********************************************************************************
; Variables and Constants
;***********************************************************************************

; CONSTANTS AND OPTIONS
#define ClockValue D'49'   ; I2C clock = Fosc/(4*(ClockValue+1))
#define LERR  4
#define LED3  3
#define LED2  2
#define LED1  1
#define LED0  0

; BANK 0 / ACCESS BANK
i2cBS  equ d'0'   ; I2C bank select
temp_SSPSTAT equ 0x000   ; contains parts of SSPSTAT, helps find I2C state
tcv_SW_status equ 0x001   ; current value of switches (high active)
rcvBuffer     equ  0x002   ; I2C RX buffer
sndBuffer     equ  0x003   ; I2C TX buffer
slave_addr    equ  0x004   ; storage for desired slave address

;***********************************************************************************
; NORMAL OPERATION
; Main Code
;***********************************************************************************

org 0x0000    ; handle reset vector
goto main

org 0x0008    ; handle high priority interrupts
goto main

org 0x0018    ; handle low priority interrupts
goto main

main
     call setup
     
outer_loop
     ; get switch status value from clicky board
     movlw      d'4'    ; select clicky
     movwf      slave_addr,0
     call  i2cStart
     call  i2cReadAddr
     call  i2cReadLast
     call  i2cStop
     call  delay
     movf      rcvBuffer,W,0
     movwf      sndBuffer,0
     iorwf      rcv_SW_status,W,0
     movwf      LATB,0
     ; write switch status to blinky board
     movlw      d'6'    ; select blinky
     movwf      slave_addr,0
     call  i2cStart
     call  i2cWriteAddr
     call  i2cWriteData
     call  i2cStop
     call  delay
     goto outer_loop

;*****************************************************************************
;=============================================================================
; FUNCTIONS
;*****************************************************************************
;*****************************************************************************
;----------------------------------------------------------------------------
; Initiate I2C start sequence
;i2cStart
;----------------------------------------------------------------------------

       bsf     SSPCON2, SEN, 0        ; initiate start condition.
       ; now make sure the start procedure is completed
       btfsc     SSPCON2, SEN, 0      ; read the bit state
       goto   $-2                     ; module busy, so wait.
       return

;----------------------------------------------------------------------------
; Initiate I2C stop sequence
;i2cStop
;----------------------------------------------------------------------------

       bsf     SSPCON2, PEN, 0       ; initiate stop condition
       ; hardware automatically clears it when done. So wait until done
       btfsc     SSPCON2, SEN, 0
goto $-2
return

; Transmit the address of slave on the I2C clock and data lines.
i2cWriteAddr

    movlb i2cBS
    movf slave_addr, W, 1    ; load address
    movwf SSPBUF, 0          ; and move into SSPBUF

    ; Check for completion of event
    btfsc SSPSTAT, R_W, 0    ; check the write bit state.
goto $-2                 ; module is busy, check again
return

; Writes Date to the slave. Move the value to be sent into sndBuffer
; before calling subroutine.
i2cWriteData

    movlb i2cBS
    movf sndBuffer, W, 1    ; load desired value
    movwf SSPBUF, 0         ; and move into SSPBUF

    ; Check for completion of event
    btfsc SSPSTAT, R_W, 0    ; check the write bit state.
goto $-2                 ; module is busy
return

; Transmits the address of slave - read mode (into the i2c line)
i2cReadAddr

    movlb i2cBS
    movf slave_addr, W, 1    ; load address
    incf WREG, W, 0          ; set Read bit
    movwf SSPBUF, 0         ; and move into SSPBUF

    ; Check for completion of event
    btfsc SSPSTAT, R_W, 0    ; check the write bit state.
goto $-2                 ; module is busy, check again
return

; Read One byte of info from the I2C line. This is not the last byte.
i2cRead

    bsf SSPCON2, RCEN, 0  ; initiate I2C read

    ; Check for event completion
    btfsc SSPCON2, RCEN, 0
goto $-2

    ; send ACK to indicate more data transmission is on the way.
call initiateAck

    movf SSPBUF, W, 0     ; move read info out of buffer
    movlb i2cBS
    movwf rcvBuffer, 1

return

; Read One byte of info from I2C line. This also indicates the master
; is done reading from the slave.
i2cReadLast

    bsf SSPCON2, RCEN, 0  ; initiate I2C read
; Check for event completion
btfsc SSPCON2, RCEN, 0
goto 5-2

; send NACK to indicate end of data receive
call initiateNack

movf SSPBUF, W, 0 ; move read info out of buffer
movib i2cBS
movwf rcvBuffer, 1
return

; Check if the I2C bus is free so we can use it, and the master is
; not in any process already.
checkIdle

; generic idle check module
btfsc SSPSTAT, R_W, 0 ; check for transmit in progress
goto 5-2 ; module busy
movf SSPCON2, W, 0 ; get a copy of SSPCON2
andlw 0x1F ; mask out non-status
btfss STATUS, Z, 0 ; test for zero state
goto $-6 ; bus is busy
; test again
return

; Initiate a NACK
initiateNack

bsf SSPCON2, ACKDT, 0 ; set the ack bit to 1
bsf SSPCON2, ACKEN, 0 ; initiate Nack
goto 5-2
return

; Initiate ACK
initiateAck

bcf SSPCON2, ACKDT, 0 ; set the ack bit to 0
bsf SSPCON2, ACKEN, 0 ; initiate ack
btfsc SSPCON2, ACKEN, 0
goto $-2
return

; Small delay. Just for kicks.
delay

i=0
while (i<D'200')
nop
i++
endw
return

;***********************************************************************************
setup ; INITIALIZATION
;***********************************************************************************
movlw b'11111111' ; setup tri-state registers
movwf TRISA, 0
movlw b'11100000'
movwf TRISB, 0
movlw b'10111111'
movwf TRISC, 0
clrf LATA,0                   ; setup registers
setf LATB,0
clrf LATC,0

clrf temp_SSPSTAT,0            ; initialize variables
movlw b'11111100'
movwf rcv_SW_status,0
clrf rcvBuffer,0
clrf sndBuffer,0
clrf slave_addr,0

movlw b'00111000'             ; setup MSSP for I2C master mode
movwf SSPCON1,0
movlw ClockValue               ; set baud rate
movwf SSPADD,0
clrf SSPSTAT,0

return

;*******************************************************************************
end
Appendix B: Clicky PIC Code

;******************************************************************************
; CLICKY.ASM
; Revision: A, 4/27/2003
; Status: complete
; Future: debug I2C interface to eliminate collisions
;******************************************************************************

; Port Usage
;******************************************************************************

; RA: [7] - N/C
; [6] - N/C
; [5] - MCLR (low active)
; [4] - LERR
; [3] - LED1
; [2] - LED0
; [1] - SW2
; [0] - SW1

; RB: [7] - programming data
; [6] - programming clock
; [5] - N/C
; [4] - I2C clock
; [3] - N/C
; [2] - N/C
; [1] - I2C data
; [0] - N/C

;******************************************************************************
; I2C Interface Notes - Clicky is an I2C Slave
;******************************************************************************

; I2C master will read SW_status register

;******************************************************************************
; Include Files
;******************************************************************************

list p = 16f819
#include <p16f819.inc>

;******************************************************************************
; Variables and Constants
;******************************************************************************

; CONSTANTS AND OPTIONS
#define LERR  4
#define LED1  3
#define LED0  2
#define SW2  1
#define SW1  0
node_address equ d'4'   ; I2C node address

; BANK 0
temp_SSPSTAT equ 0x20   ; contains parts of SSPSTAT, helps find I2C state
SW_status equ 0x21   ; current value of switches (high active)
LED_SW_status equ 0x22   ; used to control clicky LEDs
ERR_status equ 0x23   ; indicates an I2C error

;******************************************************************************
; NORMAL OPERATION
; Main Code
;******************************************************************************

org 0x0000    ; handle reset vector
goto main
org 0x0004 ; handle interrupts

main
call setup

outer_loop
movf PORTA,W ; test switches and mirror to LED0/1
andlw b'00000011'
movwf SW_status
movwf LED_SW_status
bcf STATUS,C ; clear carry flag for rotate left
rlf LED_SW_status,F
rlf LED_SW_status,W
iorwf ERR_status,W ; update LERR
movwf PORTA ; output to LEDs
goto outer_loop

setup ; INITIALIZATION

bsf STATUS,R0 ; switch to bank 1
movlw b'11111111'
movwf OSCCON
movlw b'00000111'
movwf ADCON1
movlw b'11100011' ; setup tri-state registers
movwf TRISA
movlw b'11111111'
movwf TRISB
bcf STATUS,R0 ; switch back to bank 0
movlw b'11111111' ; setup registers
movwf PORTA
clr PORTB
clr temp_SSPSTAT ; initialize variables
clr SW_status
clr LED_SW_status
movlw b'00010000'
movwf ERR_status

;-----------------------------------------------------------------------------------
; SSPCON: MSSP CONTROL REGISTER (I2C MODE) (BANK 0)
; R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0
; WCOL SSPOV SSPEN CKP SSPM3 SSPM2 SSPM1 SSPM0
;-----------------------------------------------------------------------------------
movlw b'00110110' ; setup MSSP for I2C
movwf SSPCON
bsf STATUS,R0 ; switch to bank 1
movlw node_address
movwf node_address
movwf SSPADD
clr SSPSTAT

;-----------------------------------------------------------------------------------
; INTCON REGISTER (BANKS 0,1,2,3)
; R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0
; GIE/GIEH PEIE/GIEL TMR0IE INTOIE RBIE TMR0IF INT0IF RBIF
; PERIPHERAL INTERRUPT ENABLE REGISTER 1 (PIE1) (BANK 1)
; R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0
; PSPIE(1) ADIE RCIE TXIE SSPIE CCP1IE TMR2IE TMR1IE
;-----------------------------------------------------------------------------------
bsf PIE1,SSPIE ; enable I2C interrupts
bcf STATUS,R0 ; switch back to bank 0
bsf INTCON,PEIE ; enable peripheral interrupts
clr PIR1 ; clear peripheral interrupt flags
bsf  INTCON,GIE           ; enable global interrupts

return

;******************************************************************************
handle_I2C         ; I2C ISR
;******************************************************************************
  bcf    PIR1,SSPIF
  movf   SW_status,W
  movwf   SSPBUF
  bsf    SSPCON,CKP

  btfsc  SSPCON,WCOL
  goto   WCOL_detect

  bsf    ERR_status,LERR
  bcf    STATUS,RP0
  retfie

WCOL_detect
  bcf    ERR_status,LERR
  bcf    SSPCON,WCOL
  retfie

;******************************************************************************
end
Appendix C: Blinky PIC Code

;******************************************************************************
; BLINKY.ASM
; Revision: A, 4/27/2003
; Status: complete
; Future: add PWM controlled lights for effects
;
;******************************************************************************

; Port Usage
;******************************************************************************

; RA: [7] - N/C
; [6] - N/C
; [5] - MCLR (low active)
; [4] - LERR
; [3] - LED3
; [2] - LED2
; [1] - LED1
; [0] - LED0

; RB: [7] - programming data
; [6] - programming clock
; [5] - N/C
; [4] - I2C clock
; [3] - N/C
; [2] - N/C
; [1] - I2C data
; [0] - N/C

;******************************************************************************
; I2C Interface Notes - Blinky is an I2C Slave
;******************************************************************************

; I2C master will send a 2-bit value to be displayed

;******************************************************************************
; Include Files
;******************************************************************************

list p = 16f819
#include <p16f819.inc>

;******************************************************************************
; Variables and Constants
;******************************************************************************

; CONSTANTS AND OPTIONS
#define LERR  4   ; port bits
#define LED3  3
#define LED2  2
#define LED1  1
#define LED0  0
node_address equ d'6'   ; I2C node address

; BANK 0
temp_SSPSTAT equ 0x20   ; contains parts of SSPSTAT, helps find I2C state
rcv_display equ 0x21    ; lower 2-bits are value to be displayed
pa_upper_bits equ 0x22  ; top 6-bits to be output to PORTA

;******************************************************************************
; NORMAL OPERATION
; Main Code
;******************************************************************************

org 0x0000    ; handle reset vector
goto main
org 0x0004 ; handle interrupts

main
call setup

outer_loop
movf pa_upper_bits,W ; merge current upper 6-bits (including LERR)
iorwf rcv_display,W ; output received data to LED1-0
movwf PORTA
goto outer_loop

;***********************************************************************************

setup       ; INITIALIZATION
;***********************************************************************************

bsf STATUS,RP0 ; switch to bank 1
movlw b'11111111' ; set internal oscillator to 8 MHz
movwf OSCCON
movlw b'00000111' ; set PORTA[5-0] to digital I/O
movwf ADCON1
movlw b'11100000' ; setup tri-state registers
movwf TRISA
movlw b'11111111'
movwf TRISB
bcf STATUS,RP0 ; switch back to bank 0
movlw b'11111111' ; setup registers
movwf PORTA
clrw PORTB
clrw temp_SSPSTAT ; initialize variables
clrw rcv_display
movlw b'11111100'
movwf pa_upper_bits

;--------------------------------------------------------------------------

; SSPCON: MSSP CONTROL REGISTER (I2C MODE) (BANK 0)
; R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0
; WCOL SSPOV SSPEN CKP SSPM3 SSPM2 SSPM1 SSPM0
;--------------------------------------------------------------------------

movlw b'00110110' ; setup MSSP for I2C
movwf SSPCON
bsf STATUS,RP0 ; switch to bank 1
movlw node_address
movwf SSPADD
clrw SSPSTAT

;--------------------------------------------------------------------------

; INTCON REGISTER (BANKS 0,1,2,3)
; R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-x
; GIE/GIEH PEIE/GIEL TMR0IE INTOIE RBIE TMROIF INTOIF RBIF
; PERIPHERAL INTERRUPT ENABLE REGISTER 1 (PIE1) (BANK 1)
; R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0
; PSPIE(1) ADIE RCIE TXIE SSPIE CCP1IE TMR2IE TMR1IE
;--------------------------------------------------------------------------

bsf PIE1,SSPIE ; enable I2C interrupts
bcf STATUS,RP0 ; switch back to bank 0
bsf INTCON,PEIE ; enable peripheral interrupts
clrw PIR1 ; clear peripheral interrupt flags
bsf INTCON,GIE ; enable global interrupts

return

;******************************************************************************

handle_I2C ; I2C ISR
;******************************************************************************
; State 1: I2C write, address received
; State 2: I2C write, data received

bcf PIR1, SSPIF ; clear SSP interrupt flag
bsf STATUS, RP0 ; switch to bank 1
movf SSPSTAT, W ; get value of SSPSTAT
bcf STATUS, RP0 ; switch back to bank 0
andlw b'00101101' ; get [D/nA, S, R/nW, BF] bits
movwf temp_SSPSTAT ; store in temp_SSPSTAT for checking

state_1
movlw b'00001001' ; check for [D/nA 0, S 1, R/nW 0, BF 1]
xorwf temp_SSPSTAT, W
btfss STATUS, Z
goto state_2

movf SSPBUF, W ; read SSPBUF to remove address from buffer and clear BF
bsf SSPCON, CKP ; release clock for master
bsf pa_upper_bits, LERR ; clear error condition
retfie ; returns, restoring W, STATUS, BSR

state_2
movlw b'00101001' ; check for [D/nA 1, S 1, R/nW 0, BF 1]
xorwf temp_SSPSTAT, W
btfss STATUS, Z
goto state_error

movf SSPBUF, W ; read SSPBUF to remove data from buffer and clear BF
andlw b'00000011'
movwf rcv_display ; save received data
bsf SSPCON, CKP ; release clock for master
bsf pa_upper_bits, LERR ; clear error condition
retfie ; returns, restoring W, STATUS, BSR

state_error
movf SSPBUF, W ; remove whatever is in the buffer
bsf SSPCON, CKP ; release clock for master
bcf pa_upper_bits, LED3 ; mark error condition
bcf pa_upper_bits, LERR ; returns, restoring W, STATUS, BSR

;***********************************************************************************
end