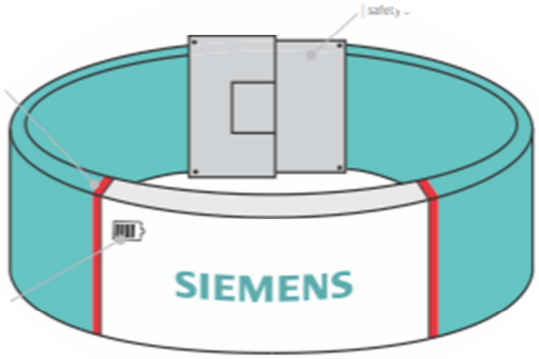
S.I.R.E.N. Wristband

Team 5

Spring 2011

Monitor	**Wristband**
Christopher Cortes	Bauyrzhan Krykpayev
Alexander Nobles	Hieu Nguyen
Steven Spinn	Stephanie Shinohara



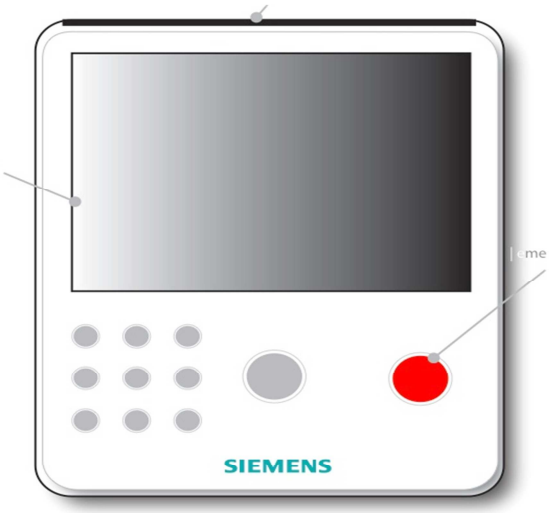


Table of Contents

Task 4

Marketing/Art Influence 5

Hardware 6

Monitor Components 6
- Monitor LCD 6
- Keypad and Keypad Encoder 6

Wristband Components 7
- Water Detection Circuit 7
- Wristband LCD 8

Shared Components 8
- Transceiver 8
- Microprocessor 9
- Voltage Regulator 10
- Batteries 10
- Miscellaneous 10

Software 11

Wireless 11
- Testing/Design 11
- Functions 12

Monitor LCD 13

Wristband LCD 14

Low Battery Detection 15

Memory Usage 16
- RAM 16
- ROM 16

Cost Analysis 17

Future Features 18

Appendix A: Schematics 20

Appendix B: Cost Breakdown 22

Appendix C: Flow Charts/State Machines 23

Monitor 23

Wristband 25

Appendix D: Source Codes 26

Table of Figures

Figure 1: *Water Detection Circuit Diagram* *7*
Figure 2: *RFM22B Wireless Transceiver Module* *9*
Figure 3: *3.3V Voltage Regulator Circuit* *10*
Figure 4: *SPI Timing for Wireless Transceiver* *11*
Figure 5: *Typical screenshot of the wristband LCD* *14*
Figure 6: *Sequence of five images of a fish swimming across the LCD display* *15*
Figure 7: *Other screenshots of LCD display* *15*
Figure 8: *Wristband PCB: Front/Back View* *19*
Figure 9: *Wristband PCB: Rotated View* *19*
Monitor Schematic *21*
Wristband Schematic *21*
Monitor: *Startup Flow Chart* *23*
Monitor: *Arm/Disarm Alarm Flow Chart* *23*
Monitor: *Set Password Flow Chart* *23*
Monitor: *Locate Wristbands Flow Chart* *24*
Monitor: *Synchronize Flow Chart* *24*
Wristband: *State Machine* *25*
Wristband: *Locate Wristbands Flow Chart* *25*

Task

Our goal for this project was to design a device that will alert the homeowner if there has been an unexpected access to the swimming pool. We as the whole team that combines students from Electrical Engineering, Marketing and Arts departments were supposed to come up with an idea how to detect this access. After the idea was established and approved, the engineering team supposed to design the prototype, assemble and test it.

Also we, as the engineering team, were required to improve on currently available on the market technologies by improving user interface, making the device more reliable, reducing the number of false alarms, and adding extra features that will be appealing to consumers.

The main requirements for the design were as follows:

1. The design should consist of two parts – external sensor and interior control panel. External sensor detects the unwanted pool access and then notifies the interior panel that alerts the homeowner.
2. Two devices should communicate via wireless link of some type. The link should be bi-directional
3. The design must be capable of being controlled by a small microcontroller
4. The design should have some type of inputs such as buttons, switches or sensors
5. The design should have outputs that will be used to notify the user such.
6. The project should have some sort of user interface.
7. All the other aspects should be determined by the EE/MKT/FA team members

Additional requirements consist of incorporating extra features that will increase the chance of our design being a commercial success while keeping costs down. Another important aspect of this project was to be able to get experience of cooperating with people from different departments in order to come up with the single design that will be working, commercially viable and aesthetically pleasing.

Marketing/Art Influence

This project was intended to be a collaborative effort between three differently-abled yet equally capable teams: an engineering group comprised of two teams of three members, a marketing team comprised of three members and a fine arts team of two members. We were meant to work through the product development process together in order to create a working prototype. Due to this reason most major design choices had to be run by the entire group in order to reach a consensus.

The marketing team had the greatest influence on the original product choice for the pool safety device. We were originally choosing between three different devices: a perimeter fence, a sonar-based device or a wristband concept. The marketing team conducted surveys on the public to determine which of these devices our target audience would be most comfortable with. The in-pool sonar device actually scored the highest during the public surveys but did no end up as our final choice. The marketing team conducted a second internal survey amongst each group. In that survey it was found that the members of the team considered the wristband device to be the least expensive, least error-prone and most effective means of pool safety. It was because of this final survey that the marketing team proposed a switch to the wristband design. The rest of the team agreed that this was a reasonable direction to take and we chose to proceed with the wristband concept.

The fine arts team provided the mock-ups and rough sketches of the product as it went through the development phase. While they had relatively little input during the decision phase they were very active and influential during the development of the wristband concept into a tangible prototype. Their designs brought forth the possibility of making this a mobile unit that the user can carry to other households or public areas as well as use within their own home. Their designs also showed that an LCD on the wristband might detract from the seriousness of the product and may make the band too bulky for use with small children. While the engineering team has currently kept the LCD in their prototype, a possible future change may be to reduce the size of the wristband or to remove it entirely.

Hardware

Monitor Components

Monitor LCD

For our project, we ordered a 24 x 2 LCD that supported character encoding. Though most project teams had initially decided to use touch screens, we agreed that we didn't need such a complex interface.

In addition to the connections to VDD/Ground and the microprocessor, we also had to modulate the power flow to the LCD's backlight. We used two potentiometers in series with the LCD's voltage inputs - one to control the voltage to the backlight, and the other to control the contrast between the letters and the background. To ensure that no power overload would occur, we also connected a resistor in series with the backlight potentiometer.

To preserve data I/O pins on the microprocessor, we chose to set up our LCD in 4-pin mode. This involved tying four of the LCD's I/O lines to ground.

Keypad and Keypad Encoder

The keypad we chose for our project (a standard 4x4 input device) had an unexpected method of sending input signals to the microprocessor: instead of feeding a 1 or a 0 to each I/O line, the keypad instead acted as a multi-way switch that bridged gaps between two of the eight I/Os.

To manage these bridged connections, we installed a keypad encoder - the EDE1144 chip - to collect information from the keypad and forward the relevant information to the microprocessor. The encoder keeps all four of the keypad's data input lines powered until it receives a signal from one of the keypad's data output lines, indicating that some button was pressed. Using a 4 MHz oscillator (separate from the system's 12 MHz oscillator for simplicity's sake), the encoder then "scans" through the various connection combinations of the keypad by powering one of the keypad's four data input lines (organized by row) and checking to see which keypad output line (organized by column) has a powered signal, if any. The encoder then sends a data valid signal (active low) to the microprocessor and asks it to read data from the encoder's four output lines (corresponding to the keypad's input lines). In addition, we also connected our encoder's beeper output to a small toggleable buzzer to give the user audio feedback, if desired.

The manufacturer of the encoder also notes that the device supports serial output, reducing the I/O lines required from four to one. However, since this would have required a dedicated serial-to-parallel device or an address system, we chose to use the parallel data connection.

Wristband Components

Water Detection Circuit

One of the important parts of the design is the water detection circuit. The main function this circuit is to basically detect the presence of water. In our designed we constructed a very simple circuit that uses water and its conductive properties to bridge the circuit. The circuit design is as follows:

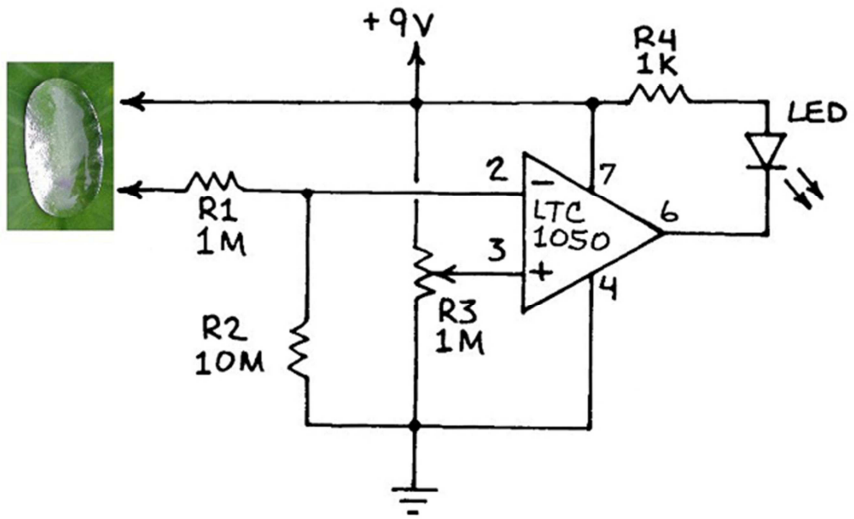


Figure 1: *Water Detection Circuit Diagram*

As it is seen from the figure the main design contains two input leads that are used as probes. Also we use an operational amplifier that is connected to potentiometer that is adjusted until LED is off. Slight modification to this circuit is that we used buzzer in series with LED to sound the alarm. Therefore when both probes are immersed in water comparator turns on the buzzer and the LED.

Instead of LTC1050 any general purpose, single-supply operational amplifier can be used. Also for our design we plan to insulate the design and locate two probes on the opposite sides of the wristband. Such arrangement will decrease the probability of accidental water spoilage on the wristband, thus decreasing the chance of alarm turning on.

Wristband LCD

One of the key features that the design team suggested was to have the wristband include an animation to entertain the child so that they want to wear it at all times. To implement this, we used an 8-bit monochromatic graphic LCD rather than the typical character LCD, allowing us to display not only text but also any type of image we want within the pixel limitation of the LCD.

The Nokia 5110 LCD is a monochromatic graphic display of 48 rows and 84 columns. It is driven by an on-board PCD8544 controller which interfaces to our microcontroller through a serial bus interface. This LCD has 8 pins: VCC connected to 3.3V, ground, and 6 control pins interfaced with Port B on the microcontroller. The backlight LEDs are software driven by one of these pins and connected in series with a resistor to limit the amount of current driving the LEDs.

Shared Components

Transceiver

The transceiver was used to provide a wireless communication interface for both the wristband and the base station monitor, allowing transmission of the alarm signal as well as a few other remote functions. We used the RFM22B ISM transceiver modules by Hope RF Electronics for prototyping our design. This particular transceiver was chosen for its operation frequency, small size, inexpensive cost, and extensive support at SparkFun. The RFM22B offers communication at the 433MHz ISM band and has an adjustable output power of up to +20 dBm, making it a good candidate for an underwater application where lower frequencies improve range of operation. With low power consumption and an operating voltage of 3.3V, the RFM22B is an ideal solution for our battery-powered wristband. More specifically, it consumes 800µA in “ready mode” and as low as 15nA in the ultra-low-power “shutdown mode.” Communication with the microcontroller was done via the GPIO pins instead of the standard 4-wire SPI interface on the transceiver; thus, the SPI protocol was implemented in software. Additional features include three configurable GPIO pins, one of which can output a 4MHz clock signal for the microcontroller, and an 8-bit ADC that can be used to monitor the voltage level for low-battery detection. It also features RX and TX FIFO’s, which were useful for encryption of data packets. The RFM22B comes in an SMD package (16mm x 16mm) and is supported by SparkFun Electronics, offering a breakout board for prototyping, sample code, and a wide user-base of experienced electronics hobbyists.

Figure 2: *RFM22B Wireless Transceiver Module*

Microprocessor

We chose to keep the original course-designated microprocessor - the MC908JL16 - as the core of our project. With the scope of our work being limited to a few kilobytes of memory data and two dozen I/O lines, there was no reason to work outside the bounds of what we were given. The JL16 is inexpensive, comes in a 28-pin DIP package for easy prototype integration, and is easily programmed using the free CodeWarrior IDE.

The main issue with the microprocessor was not the overall count of I/O pins - rather, it was the diversity of I/O pins available for certain tasks. The JL16 comes equipped with two serial I/O pins and a handful of LED sinking-capable pins. Often, the arrangement of these pins were non-consecutive. As a result, we had to assign multiple-pin device connections to non-consecutive ports, requiring us to use concatenation in our programming.

For the wristband, we implemented the JL16 in its low voltage configuration, running on 3.3V with a 4MHz clock to minimize power consumption. The GPIO pins were used to interface to several slave devices including a graphic LCD, a wireless transceiver, a water detection circuit, and several debugging buttons/LEDs. The serial communication protocols for the LCD and transceiver were programmed in software, and keyboard interrupts on Port A were used for the water detection circuit for interrupt-driven operation.

Voltage Regulator

Because the wireless transceiver required a lower voltage for operation, we needed to use a voltage regulator to turn the 5V input into a 3.3V output. This 3.3V output was also utilized by several other components in the wristband circuit for low power consumption. However, the wristband design still required a 5V signal for an op-amp in the water detection circuit. The voltage regulator we used was the LM3940, and it required two capacitors connected in parallel to the input and output. This setup is shown in Fig. 3

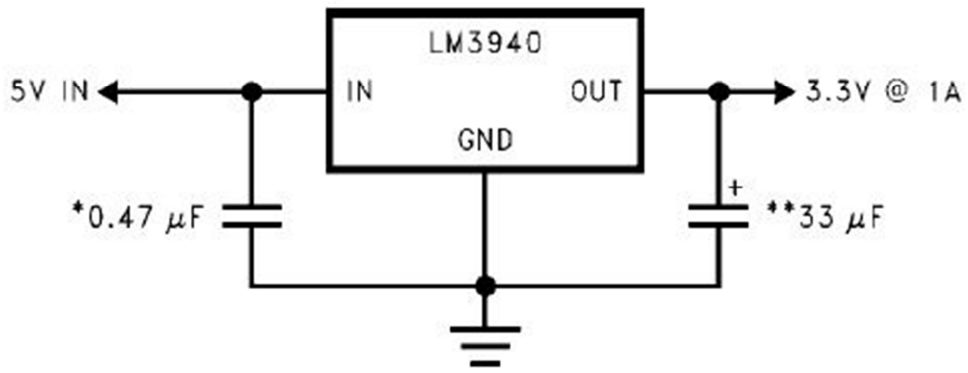


Figure 3: *3.3V Voltage Regulator Circuit*

Batteries

The batteries used in the wristband prototype were the CR2354 Panasonic 3V lithium coin-cell batteries, rated with a 560mAh capacity. A 23mm diameter battery holder was used to interface the battery to the circuit board. Using a power supply, we measured the current consumption of the wristband circuit at its most demanding state, which is when the transceiver is outputting data in transmit mode. This value averaged at 40mA. To estimate battery life, we divide the capacity by the current consumption, and multiply by a factor of 0.7 to account for conditions (like temperature) that affect the battery capacity. Thus, the estimated battery life is 560mAh / 40mA * .7 = 9.8 hours. Of course, this is assuming the wristband is constantly transmitting a signal. To implement rechargeable batteries, we may require a Coulomb counting IC to monitor the battery level, as voltage does not have a linear decay rate in some types of batteries.

Miscellaneous

The wristband prototype design features some additional parts that were used mostly for debugging, but were also incorporated into our final design. Three LEDs were used to output the status of the water detection sensor, and for debugging our software functions. One push-button switch was used for the wireless SYNC function and for interaction with the LCD. A buzzer was used on both the base station monitor and wristband systems to create an audible alarm sound, and for a locate-wristband function. A Schmitt trigger was also used in the prototype to clean up clock signal output from the crystal oscillators.

Software

Wireless

Testing/Design

The wireless transceiver uses a standard 4-wire SPI interface to communicate with the microcontroller. Because the JL16 does not have dedicated SPI pins, the serial protocol was implemented in software. This entailed writing low-level software functions to perform reading and writing of the transceiver's registers, which involved controlling the clock, data, and device select pins. Each SPI transaction was a 16-bit sequence to control registers, change the operation mode, and send/receive data packets. Because our system ran at a slower clock of 4MHz, it was easy to meet timing requirements for the SPI signals. A sample sequence is shown in Fig. 4. Because we did not have a terminal display interface to a serial output, we were unable to verify the functionality of our transceiver functions. Thus, we used a series of eight LEDs to output a byte of binary data for debugging. After the basic readByte() and writeByte() functions were working, we utilized the sample code provided by SparkFun to develop our more complex transceiver functions, which included device initialization, interrupt control, and the send/receive functions.

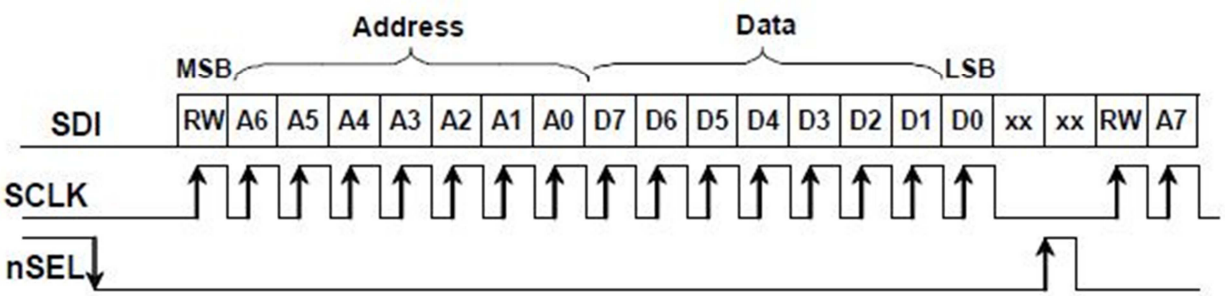


Figure 4: *SPI Timing for Wireless Transceiver*

Functions

Sync

Our Sync function allows the wristband and monitor to communicate with each other. If the two devices are not synchronized, then any action that the wristband does will not be acted upon by the monitor, and vice-versa. In order to sync the two devices, the user must select the SYNC option from the menu on the monitor. The monitor will instruct the user to press and hold the button on the wristband to begin syncing. Both the monitor and wristband will display a message if the sync is successful. If the sync is unsuccessful, the wristband will display a failure message after about 10(?) seconds and return to the main screen. To cancel the sync on the monitor, press the CLR button to go back to the main menu.

Our synchronization function is achieved through a series of handshaking between the wristband and the monitor. In its idle states, both the wristband and the monitor keep their respective transceivers in RX (receive) mode. When the monitor enters the sync function, it begins listening for a sync packet from any wristband. This is denoted by the first byte of our packet, which denotes the type of packet the device is sending/receiving. When the wristband enters syncing mode, it switches the transceiver into TX (transmit) mode. It then sends out a sync initiate packet, denoted by a 0x27H(?) in the first byte. The next 3 bytes of this packet contain the wristband's unique ID code. When the monitor receives the packet, it checks the first byte for the sync initiate packet type. It then reads the next 3 bytes for the wristband's ID and stores it into a new entry in its synchronized wristbands list. The monitor then sends out a packet back to the wristband with its own monitor ID, denoted by 0x26H. The wristband only holds one monitor ID, and thus is only able to sync with one monitor at a time. Once the wristband receives the monitor ID, it sends out a final acknowledge packet (0x25H), and the synchronization is complete. When the monitor receives the ACK packet, it displays a message that the wristband has been successfully synced. Now the user will be able to use the alarm and locate features between the monitor and wristband.

Locate

The Locate function on the monitor is designed to allow the user to find a misplaced wristband by emitting a continuous beep from the wristband. When the user selects the Locate function in the monitor's menu, all of the wristbands the monitor has synced with will begin to beep. The user can stop the wristbands by pressing the ENTER key on the monitor or by pushing the button on the wristband. Stopping the Locate function through the monitor will stop all wristbands simultaneously, whereas pushing the wristband button will only stop that individual wristband.

When the user selects the Locate function on the monitor, it will switch its transceiver into TX mode and begin transmitting a locate packet, denoted by a 0x29 in the packet header. When the wristband receives a locate packet, it will compare the next 3 bytes to its stored monitor ID value. If the values match, it will begin to beep. If the values do not match, it will discard the packet and do nothing. The wristband will continue to beep until the monitor sends a stop packet, denoted by a 0x28H, or until the user presses the button on the wristband. If the user stops the wristband with the button and then sends out a stop packet from the monitor, the packet will be ignored and dropped.

Alarm

As the most important part of the system, the alarm function is enabled when the user selects “Enable Alarm” from the monitor’s menu. When the alarm is enabled, the wristband will send out an alarm packet on the transceiver if it detects water on its water detection circuit. When the monitor receives the alarm, it will display a warning message and set off the buzzer.

When the wristband detects water, the hardware triggers an interrupt in the software and switches the transceiver to TX mode. It then sends out an alarm packet, denoted by a 0x30H in the packet header. When the monitor receives the alarm packet, it checks its wristband list for a match on the ID bytes. If they match, the alarm is triggered. The user must enter their password to disable the system, which will stop the alarm.

Monitor LCD

Our 24x2 LCD controller conforms to the Hitachi HD44780 standard, so interfacing with the software is fairly simple. The LCD is used in a parallel interface using four outputs for the data lines. Since our LCD is an 8-bit display, we output 4-bit data twice in order to display our data. The controller accepts ASCII input, so our software simply outputs each character in hex and writes to the address on the LCD for specific positions. For example, to write to the first character on the first line, the software writes to address 0x20H with its character data. The second line starts at 0x40H. Since our LCD only has 12 characters on each line, addresses 0x32H to 0x39H are unused. To see more information on how our software controls the LCD, look up the strout(), cmdout(), and nibout() functions in the source code, located in the Appendix at the end.

Wristband LCD

Although the Nokia graphic LCD gave us more flexibility in what we could display, it also provided another challenge to learn how to write to it since no previous team has used this particular LCD before. After running and examining the sample code provided on the supplier's website, we were able to understand the writing process to the display. From there, images were drawn for various functions and those bitmaps were converted to 8-bit pixel arrays for the LCD.

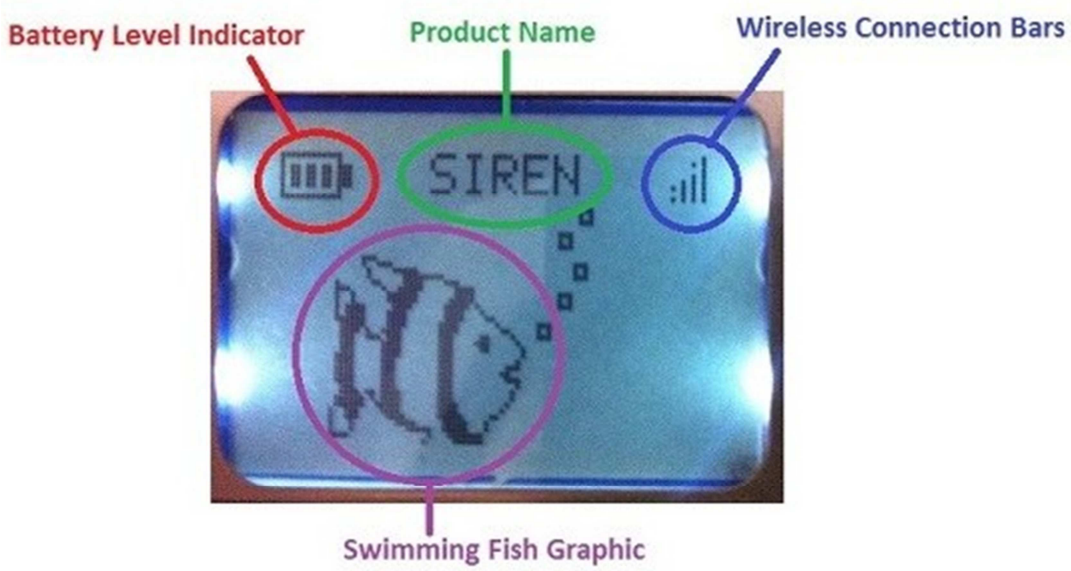


Figure 5: *Typical screenshot of the wristband LCD*

The main components of each screen are shown in Fig. 5 above. On the top line of every screen is the same set of indicators. The battery level is shown on the top-left corner and the bars within the battery icon represent how much voltage is being provided by the battery. At the top-center of the display is the name of the product, SIREN. To the top-right corner are wireless connectivity level bars which will be updated according to the signal strength between the monitor and wristband. In this prototype, however, the bars are just there for reference and do not actually represent signal strength. During normal functions, the wristband will show an image of a fish as it appears to swim across the screen as shown in Fig. 6 below. Here we implemented a simple sequence of five screens that run in a loop. If put to market, there is room for to add more complex interactions with the fish but for prototyping purposes these five screens demonstrate a simple representation of the animation.

Figure 6: *Sequence of five images of a fish swimming across the LCD display*

Fig. 7 shows the remaining screens implemented in our wristband design. The splash screen shows the Siemens logo on start up. When put in synchronizing mode, the LCD displays animated boxes that continue to appear across the screen as the wristband is in the process of synchronizing with a monitor. This animation cycles a few times and then the LCD notifies the user if it was successful or if it was unable to make a connection within a certain amount of time.

Figure 7: *Other screenshots of LCD display*

Low Battery Detection

Currently we are using two 5V Lithium batteries that provide the circuit with approximately 560 mAh whereas our design currently draws approximately 40 mA. Thus, the battery life of the wristband is approximately 10-12 hours. For the current purpose such battery life is enough, but we would like to extend it in future by using other circuit elements that require less power consumption. Another important moment is that almost whole circuit except op amps can operate on 3V batteries. Changing the circuit, so all its elements can operate on 3V will increase the battery life.

In order to measure the battery level we designed the low battery detection program using in-built functions of the receiver. Originally we tried to use microcontroller and design the low battery detection circuit. But this design requires additional elements, thus it will increase the design size.

Low battery detection(LBD) circuit is already built into the transceiver. We constantly poll the LBD output that goes through analog to digital converter (ADC) in order to detect correct battery voltage.

$$BatteryLevel = 1.7 + 50mV * ADCvalue$$

This formula is used to calculate the battery level where 1.7V is the minimum operating voltage of the transceiver; 50mV is the LSB step size of the ADC. Thus, depending on the battery level, we update the wristband screen to demonstrate new battery level.

Memory Usage

RAM

The software for the monitor utilized 194 bytes of RAM, out of the total 512 bytes of on-chip random-access memory. The wristband used 174 out of 512 bytes of the RAM on the microprocessor. This is approximately a 38% usage for the monitor and a 34% usage for the wristband, and leaves a considerable amount of space for additional features

ROM

The software for the monitor utilized 4296 bytes of the available 16KB of FLASH memory for program storage. The wristband used 6438 bytes out of the 16KB of FLASH memory. This is approximately a 26% usage for the monitor and a 40% usage for the wristband. Most of the wristband ROM was used by images for the LCD, which were declared as constant character arrays. The extra space in read-only memory can be utilized for additional features such as making the graphic LCD more visually interactive.

Cost Analysis

A key design goal the engineering team had was to minimize the cost of a production unit of the device. As this is intended to be Siemens' first foray into the pool safety market it was determined that the initial startup cost for the company should be as minimal as possible. A price point of $350.00 was chosen during the design process in order to position the product on the low end of the pool safety device spectrum. The marketing team felt this would allow Siemens to minimize risk and to open the product to a larger target audience. A detailed breakdown of the costs can be found in the appendix.

A major expense for the monitor prototype was the LCD. Rather than go with a more expensive color graphic LCD we chose to use a relatively small character LCD. Looking at how inexpensive the bulk cost of graphic LCDs are and the amount of IO pins remaining on the microcontroller a future iteration of the monitor could possibly use a graphic LCD to display more information. Another cost decision that had to be made for the monitor was the keypad. A 4x4 keypad was less expensive than a 3x3 numerical keypad and extra push buttons. It also required fewer IO pins on the microcontroller. This alternative doesn't provide very good button placement and, while useful for the prototype, the shippable design would require a 3x3 keypad with several push buttons that can be placed around the monitor device.

The most important cost decision regarding the wristband had to do with the LCD as well. In the original design it was decided that a graphic LCD would be an important interface device and that it should be kept. As the fine arts group progressed in their designs it was found that the rest of the team was willing to remove the LCD in order to decrease the size of the wristband. The marketing team also pointed out that the removal of the wristband would decrease the cost of each wristband by almost half. While the graphic LCD was kept for the prototype, in a future shippable version it could be removed due to cost issues. The other important cost decision for the wristband was the use of a certain wireless transmitter. Rather than use the XBee model that was offered at the beginning of the class, we chose to find a less expensive alternative. Our alternative, the RFM22B-S2 is less expensive than the XBee model and offered a key feature that the wristband needed to function: a battery meter. While the learning curve was somewhat steeper for this wireless transceiver than the XBee, the RFM22B-S2 suited the projects' needs very well and is packaged in a smaller form factor.

Future Features

This prototype includes all of the important features of the final design; however, there are a few additional components that have not been implemented yet. According to the marketing surveys and design suggestions both the wristbands and receiver monitor unit should be portable in case the user wishes to use them for travel purposes. In order to work toward the final working product, engineering teams should add rechargeable batteries to the wristband and the monitor, wireless connectivity level indicators, a heartbeat signal between units, and any other additional feature necessary for this product.

Rechargeable batteries allow the user to take the product with them outside of the restrictions of the house. While mounted on the wall, the monitor can be plugged in to a power source; however, it should also be able to run off of rechargeable batteries in case the user wants to travel with the product. There should also be functions on the monitor to check the battery level of the wristbands in the absence of an LCD on the wristband. The fact that this product uses rechargeable batteries also has good marketing value. Using rechargeable batteries is extremely cost-effective for the user and it also makes the product eco-friendly.

Wireless connectivity indicators will be helpful for the user to know how far away the wristband can get from the monitor before it can no longer communicate properly. It will also allow the user to visually check that it is linked properly and there are no problems with the connection.

A **heartbeat signal** should be sent out by the wristband to the monitor to ensure that everything on the wristband is working properly. This would make sure that the battery on the wristband is properly charged and that the wristband is within wireless range of the monitor. Including a heartbeat signal would also solve any confusion of the transceiver's capability to transmit the emergency signal through water. The updated design with a heartbeat signal will trigger the monitor's alarm (if it is armed) when the wristband in submerged in water or if it is out of wireless range. In both cases the "heartbeat" will stop and the monitor will sound the alarm.

After completing the hardware configuration for the wristband prototype, we wanted to visualize the approximate dimensions of the electronics components of an actual manufactured product. To do this, we looked for the smallest package of each component, and removed the unnecessary components from the prototype design (such as debugging LEDs, dedicated crystal oscillator, etc.). Then we imported this

information into Altium Designer (which was used to create the schematic diagram) to design a **printed circuit board** (PCB). A 3-dimensional model was created for each major component to visualize thickness. The board layout put the LCD on one side of the PCB and the rest of the electrical components on the other side. As you can see in Fig. 8, the PCB is really limited by the size of the LCD, and thus, the final design for manufacturing can be made smaller.

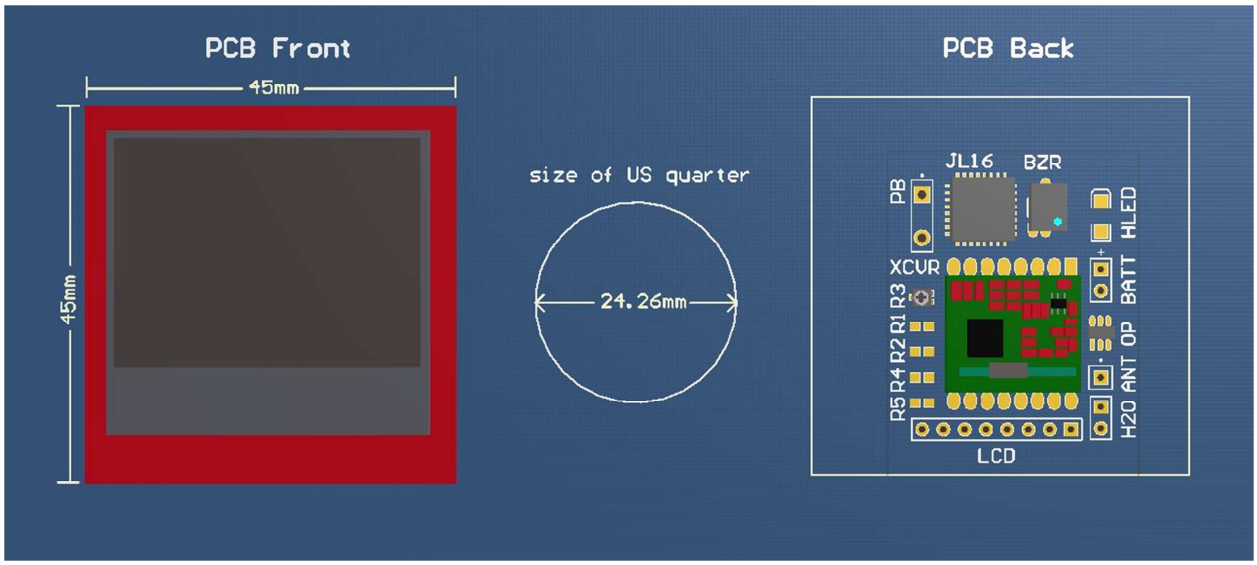
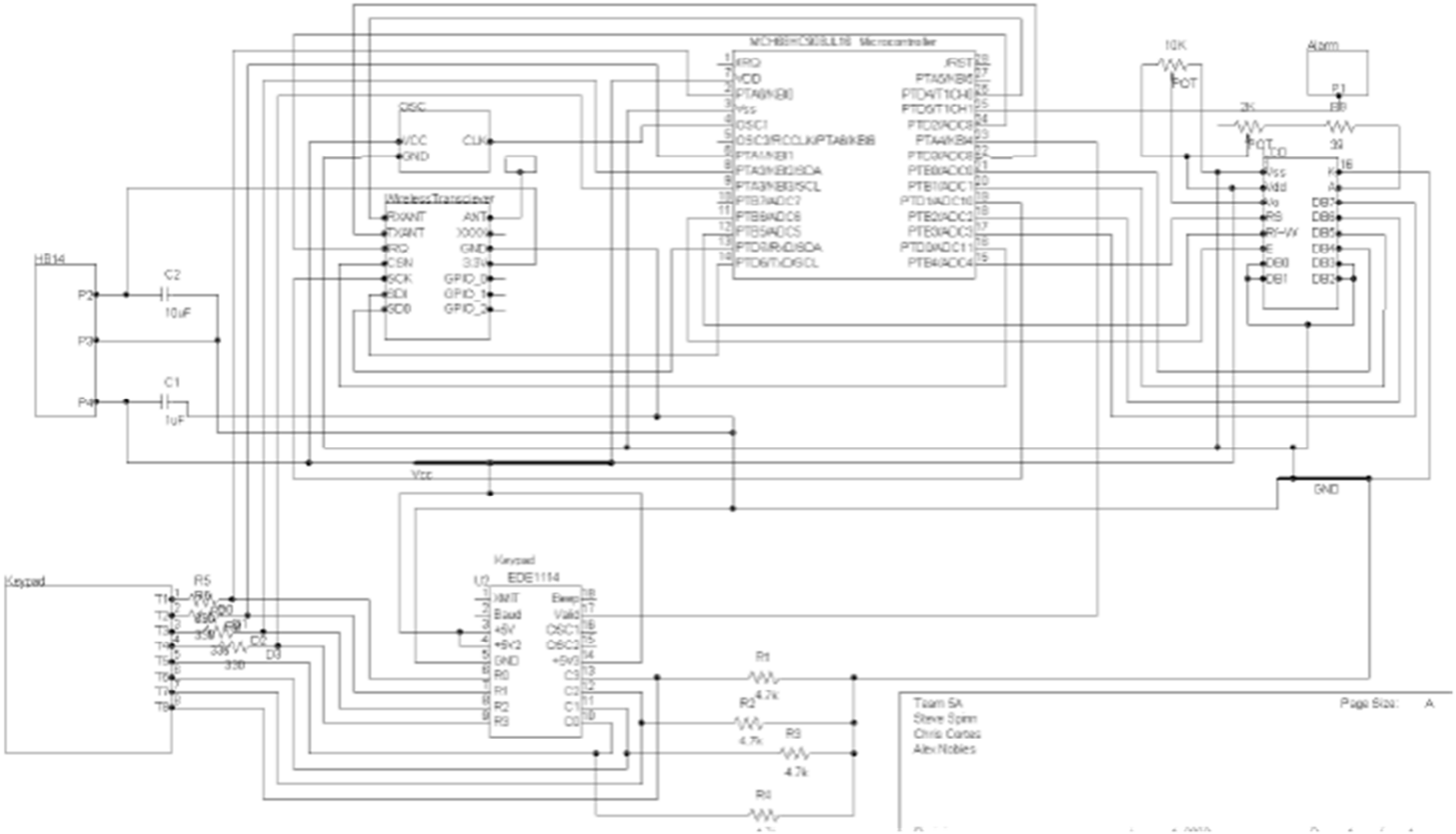


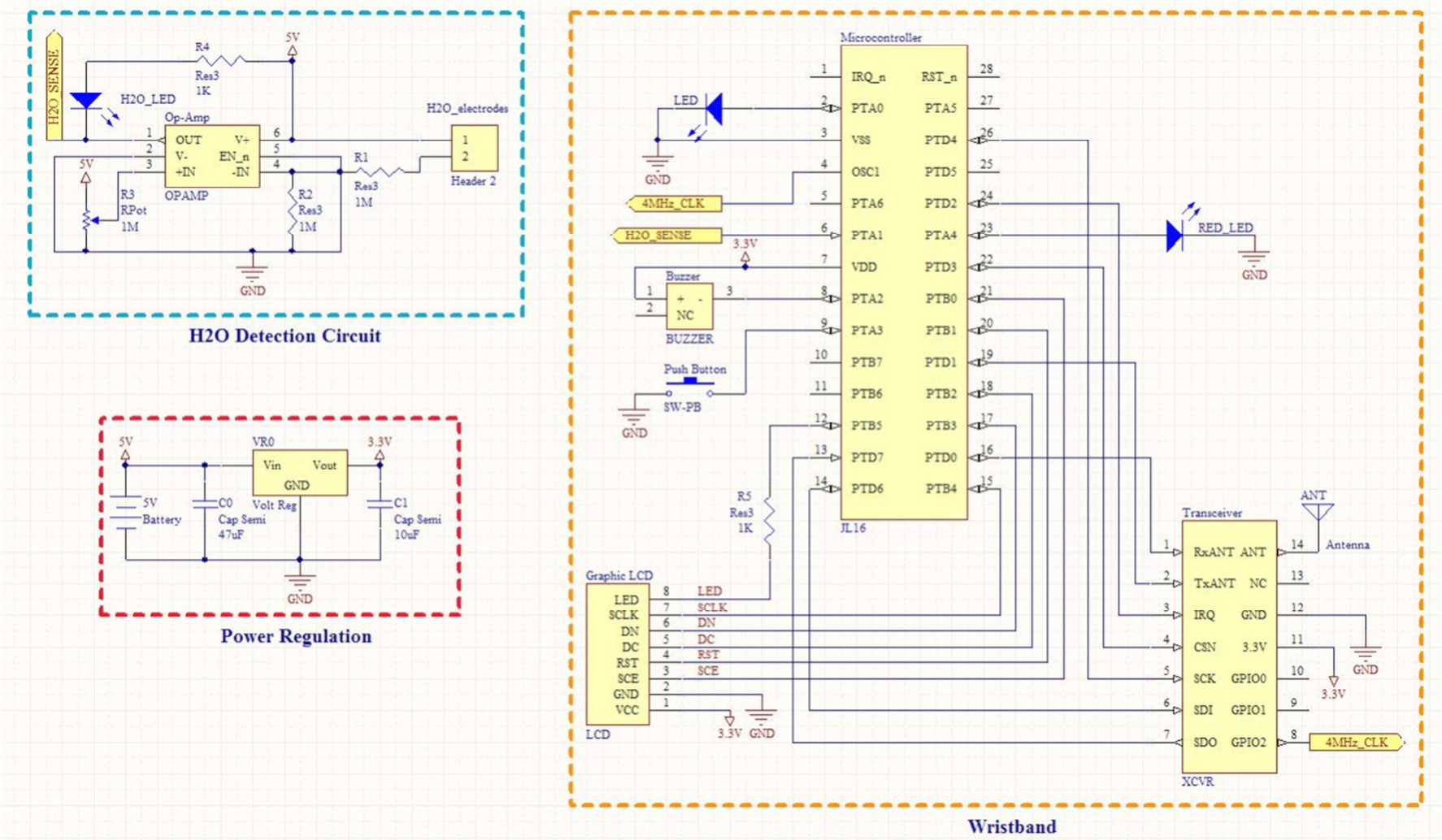
Figure 8: *Wristband PCB: Front/Back View*

Figure 9: *Wristband PCB: Rotated View*

Appendix A: Schematics



Monitor Schematic



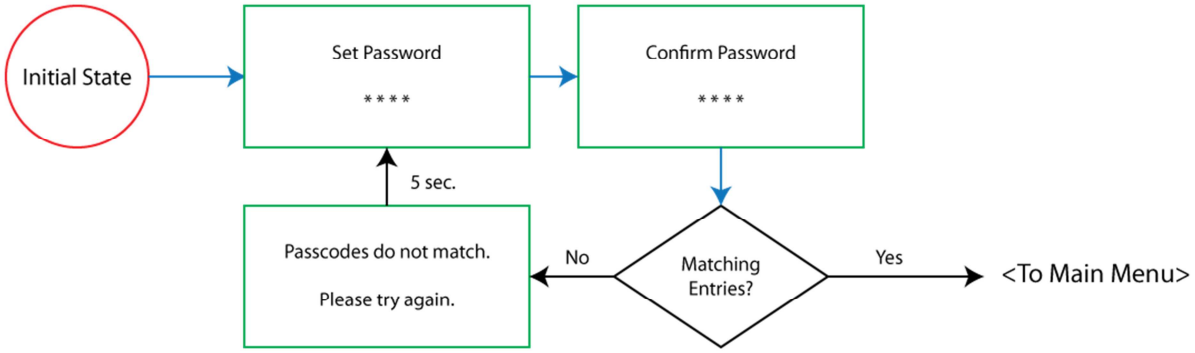
Wristband Schematic

Appendix B: Cost Breakdown

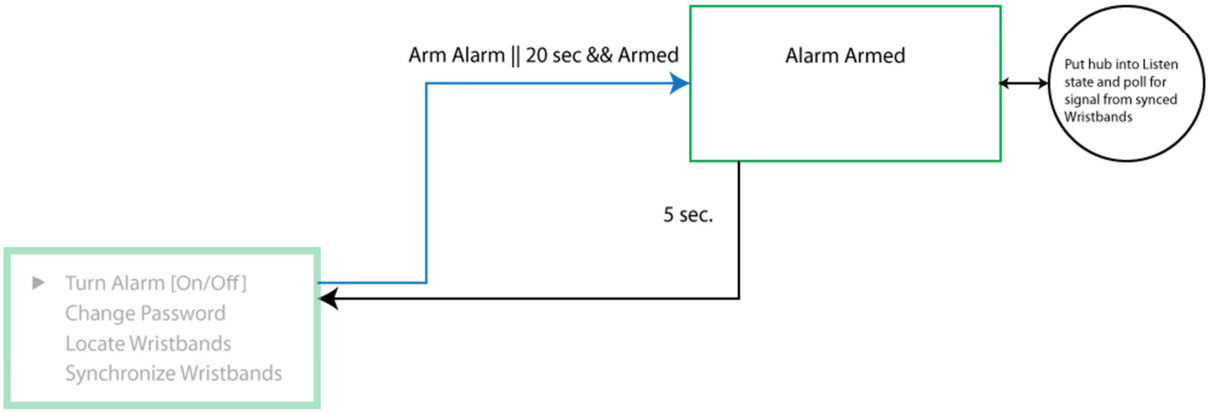
Part	Model	Prototype Cost	Bulk Cost
Microcontroller	MC908JL16	$ 3.00	$ 1.74
Transceiver	RFM22B-S2	$ 11.95	$ 9.56
Buzzer	76dB Piezo	$ 3.49	$ 3.49
24x2 LCD	CFAH2402A-NYG-JP	$ 22.98	$ 6.96
4x4 Keypad	GH5003-ND	$ 14.34	$ 10.76
Miscellaneous Parts		$ 1.00	$ 1.00
TOTAL (Monitor):		**$ 56.76**	**$ 33.51**
Microcontroller	MC908JL16	$ 3.00	$ 1.74
Transceiver	RFM22B-S2	$ 11.95	$ 9.56
Buzzer	KBS-13DA-12A	$ 3.49	$ 0.25
Nokia Graphic LCD	LCD-10168	$ 9.95	$ 7.96
Battery (x2)	CR2032	$ 0.56	$ 0.56
Miscellaneous Parts		$ 1.00	$ 1.00
TOTAL (Wristband):		**$ 29.95**	**$ 21.07**

Appendix C: Flow Charts/State Machines

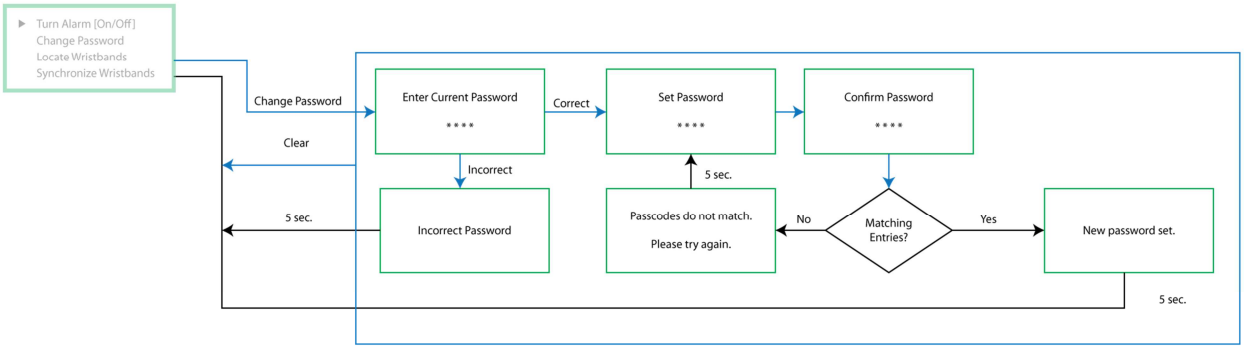
Monitor



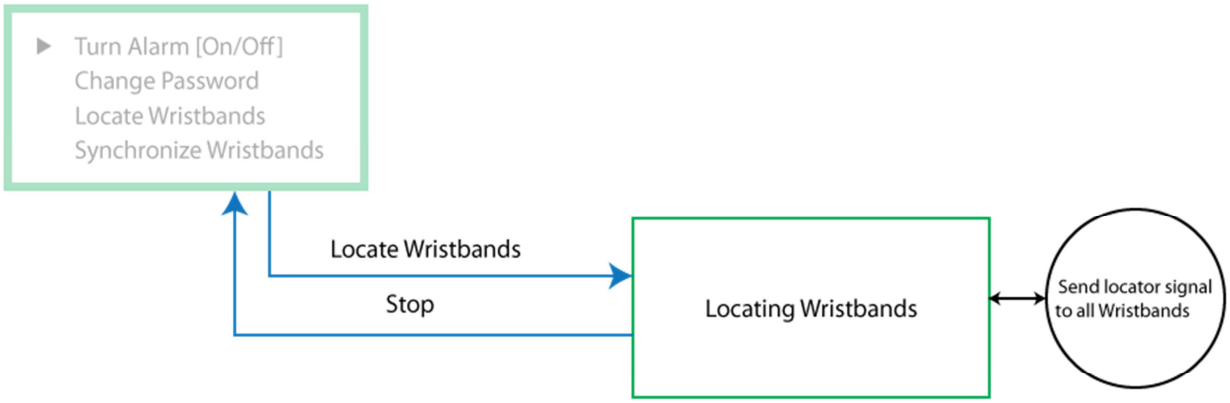
Monitor: *Startup Flow Chart*



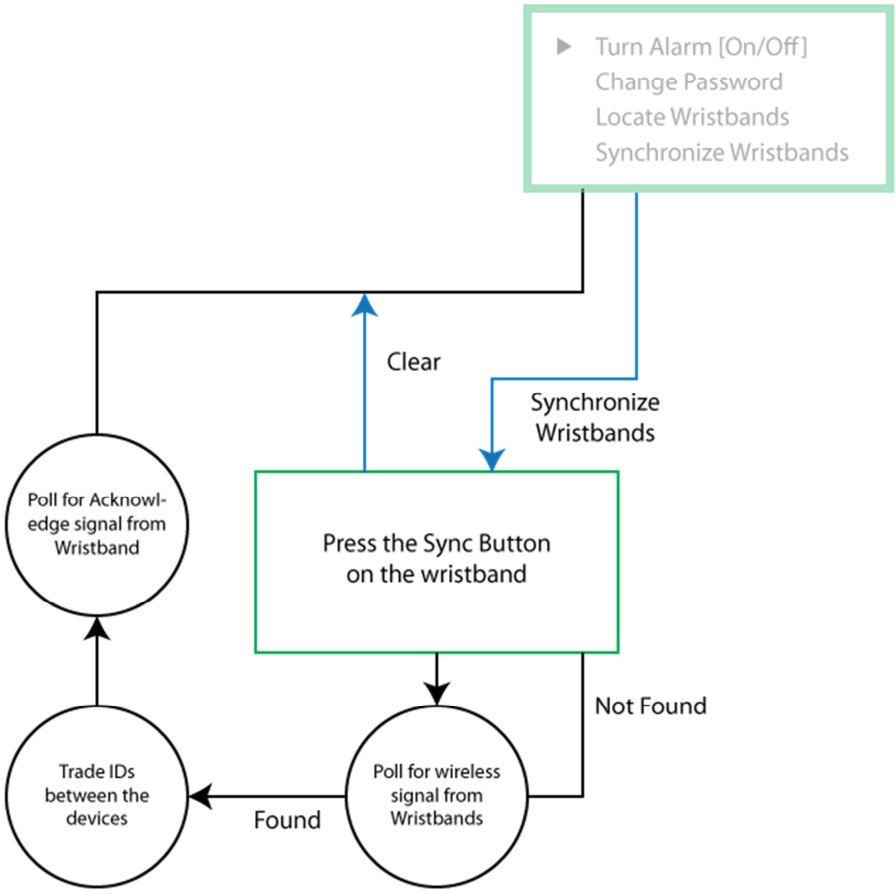
Monitor: *Arm/Disarm Alarm Flow Chart*



Monitor: *Set Password Flow Chart*

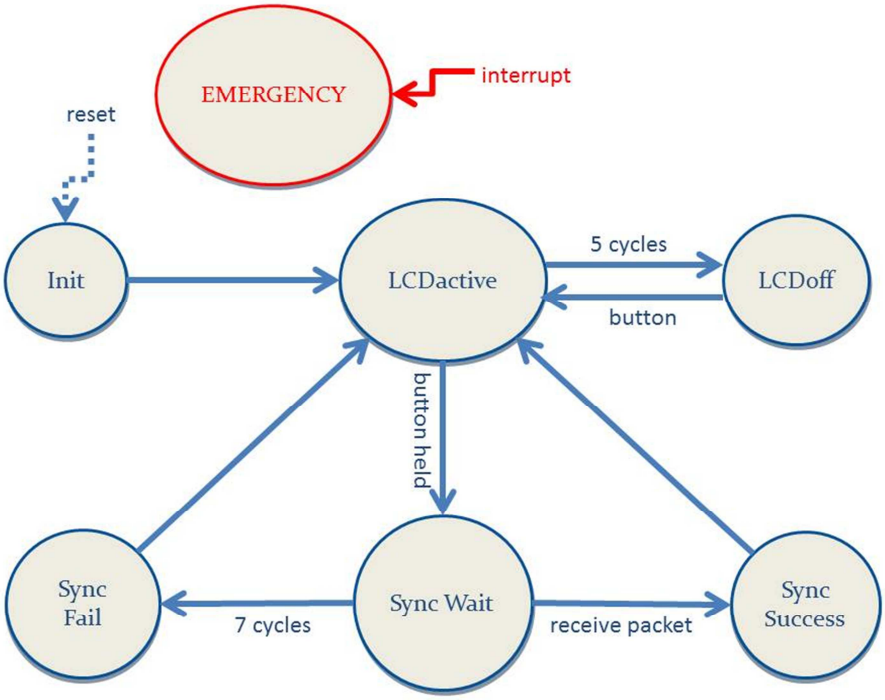


Monitor: *Locate Wristbands Flow Chart*

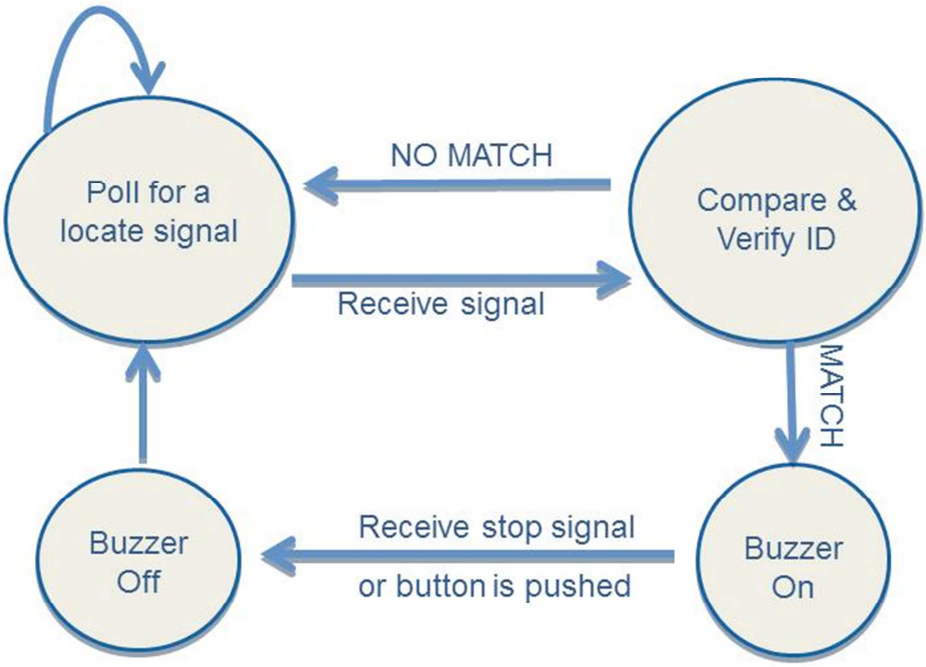


Monitor: *Synchronize Flow Chart*

Wristband



Wristband: *State Machine*



Wristband: *Locate Wristbands Flow Chart*

Appendix D: Source Codes

```c
/*************************************************************
*       CWJL16-5 - Demonstrate interface to a parallel LCD display
*
*       This program will print a message on an LCD display
*       using the 4-bit wide interface method
*
*       Port B, bit 4 () - output to RS (Register Select) input of display
*               bit 5 () - output to R/W (Read/Write) input of display
*               bit 6 () - output to E (Enable) input of display
*       Port B, bits 0-3 - Outputs to DB4-DB7 inputs of display.
*
*       The second line of the display starts at address 0x40.
*
*************************************************************/

#include <hidef.h> /* for EnableInterrupts macro */
#include <string.h>
#include "derivative.h" /* include peripheral declarations */

/*
  The following puts the dummy interrupt service routine at
  location MY_ISR_ROM which is defined in the PRM file as
  the start of the FLASH ROM
*/

#pragma CODE_SEG MY_ISR_ROM

#pragma TRAP_PROC
void dummyISR(void)
{
}

/*
  This pragma sets the code storage back to default area of ROM as defined in
  the PRM file.
*/

#pragma CODE_SEG DEFAULT

typedef unsigned char uchar;
typedef unsigned int uint;

#define XCVR_RxANT    PTD_PTD4
#define XCVR_TxANT    PTD_PTD3
#define XCVR_IRQ      PTD_PTD2
#define XCVR_CSN      PTD_PTD0
#define XCVR_SCK      PTD_PTD1
#define XCVR_SDI      PTD_PTD6
#define XCVR_SDO      PTD_PTD7

//Global Variables
uchar locate[17] =     {0x29, 0x12, 0x34, 0x56, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B,
0x3C, 0x3D, 0x3E, 0x3F, 0x78};
```



```c
uchar doneLocate[17] = {0x28, 0x12, 0x34, 0x56, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B,
0x3C, 0x3D, 0x3E, 0x3F, 0x78};
uchar sendSyncID[17] = {0x26, 0x12, 0x34, 0x56, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B,
0x3C, 0x3D, 0x3E, 0x3F, 0x78};

uchar RF_RXBUF[35];
uchar syncList[6][3];
unsigned char one = 1;
void initialize(void);
void strout(int, unsigned char *);
void iniout(unsigned char);
void cmdout(unsigned char);
void datout(unsigned char);
void nibout(unsigned char);
void busywt(void);
void del1m(int);
void del40u(int);
unsigned char checkKeypadInput(void);
void lcd_write(int, unsigned char*);
void clearLCD(void);
void setPassword(void);
unsigned char enterPwd(void);
void waitForDelay(void);
void waitForEnter(void);
void syncWristbands(void);
void locateWristbands(void);
void armSystem(void);
void disarmSystem(void);
void checkAlarm(void);
void write0(void);
void write1(void);
void writeByte(char address, char data);
char readByte(char address);
void init_RFM22(void);
void to_tx_mode(char[], char);
void to_rx_mode(void);
void rx_reset(void);
void send_read_address(uchar addr);
uchar send_command(void);

volatile struct {
  unsigned char Keypad:4;
} MyPTA @0x0000;

#define Keypad MyPTA.Keypad
#define KEYPAD_VALID !PTA_PTA4
#define LCD1 0
#define LCD2 0x40
#define ALARM_BUZZER PTA_PTA5

/*
  This pragma sets the area for allocating variables to the zero page
  of RAM (0x00 to 0xff) for more efficient access.
```



```c
*/

#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE

unsigned char delay0;
unsigned char delay1;
unsigned char delay2;


unsigned char pwd[4];
unsigned char armed;
unsigned char state; //State 0 -> Initial state - set password
                     //State 1 -> Menu - Arm/Disarm
                     //State 2 -> Menu - Change Pwd
                     //State 3 -> Menu - Sync Wristband
                     //State 4 -> Menu - Locate Wristband
                     //State 5 -> Enter Pwd - *Not used*
                     //State 6 -> Change Pwd - *Not used*
                     //State 7 -> Status

                     //Current Keypad Configuration
                     // * -> Left
                     // # -> Right
                     // A -> Menu
                     // B -> CLR
                     // C -> Bksp
                     // D -> Enter

unsigned char alarmActive;




/*
  This pragma sets the area for allocating variables back to the default area
  of RAM as defined in the PRM file.
*/

#pragma DATA_SEG DEFAULT

/*
    Declare these strings as "const" so they are allocated in ROM
*/




const unsigned char statusHdr[] = "PROTECTION STATUS:";
const unsigned char statusEn[] = "<< ENABLED >>";
const unsigned char statusDis[] = "<< DISABLED >>";
const unsigned char menuHdr[] = "Main Menu";
const unsigned char menuDisable[] = "> Disable Alarm";
const unsigned char menuEnable[] = "> Enable Alarm";
```



```c
const unsigned char menuSetPwd[] = "> Set Password";
const unsigned char menuChangePwd[] = "> Change Password";
const unsigned char menuLocate[] = "> Locate Wristband";
const unsigned char menuSync[] = "> Sync Wristband";

const unsigned char setPwdHdr[] = "Set 4 Digit Password";
const unsigned char reenterPwdHdr[] = "Re-enter to confirm:";
const unsigned char enterPwdHdr[] = "Enter Password";


const unsigned char locateHdr[] =  "Locating Wristbands";
const unsigned char locateStop[] = "> Stop";

const unsigned char syncHdr[] =  "Press Sync button on";
const unsigned char syncInstr[] = "wristband to sync.";

const unsigned char pwdChanged[] = "Password changed";
const unsigned char pwdChanged2[] = "successfully.";
const unsigned char pwdSet[] = "Password set";
const unsigned char pwdMismatch[] = "Password mismatch.";
const unsigned char tryAgain[] = "Try again.";

const unsigned char incorrectPwd[] = "Incorrect password";
const unsigned char pwdPrompt[] = ">";

const unsigned char systemArmed[] =  "System Armed";
const unsigned char systemDisarmed[] =  "System Disarmed";

const unsigned char syncSuccessHdr[] = "Wristband Synced:";
const unsigned char syncFailureHdr[] = "Sync Failure";



void armSystem()
{
    armed = 1;
    lcd_write(LCD1, (unsigned char *)systemArmed);
    waitForDelay();
}

void disarmSystem()
{
    ALARM_BUZZER = 1;
    alarmActive = 0;
    armed = 0;
    lcd_write(LCD1, (unsigned char *)systemDisarmed);
    waitForDelay();
}

void lcd_write(int addr, unsigned char* c) {
  clearLCD();
  strout(addr, c);
}
```

```c
void syncWristbands()
{
  unsigned char key;
  unsigned char i, j;
  unsigned char success = 0;
  unsigned char bandID;
    lcd_write(LCD1, (unsigned char *)syncHdr);
    strout(LCD2, (unsigned char *)syncInstr);
    //WIRELESS CODE GOES HERE -> WAIT TO RECEIVE SYNC INFO -> If CLR pressed, cancel
    while(one) {
      if(KEYPAD_VALID) {
        key = checkKeypadInput();
        if(key == 'B') {
          rx_reset();
          lcd_write(LCD1, "Sync cancelled.");
          waitForDelay();
          return;
        }
      }
      if(XCVR_IRQ == 0)
    {
        send_read_address(0x7F);
        for (i=0 ; i<17 ; i++)
            RF_RXBUF[i] = send_command();

        if(RF_RXBUF[0] == 0x27)
        {
          lcd_write(LCD1, "Sync request received.");
          strout(LCD2, "Syncing...");
          for(i = 0; i < 6; i++)
          {
            if(syncList[i][0] == RF_RXBUF[1] && syncList[i][1] == RF_RXBUF[2] && syncList[i][3]
            == RF_RXBUF[3])
            {
              lcd_write(LCD1, "Wristband already");
              strout(LCD2, "synced.");
              return;
            }
          }
          for(i = 0; i < 6; i++)
          {
            if(syncList[i][0] == 0x00)
            {
              bandID = i;
              for(j = 0; j < 3; j++)
              {
                syncList[i][j] = RF_RXBUF[j+1];
              }
              success = 1;
              break;
            }
          }
```



```c
          //add next 3 bytes to sync list
          rx_reset();
          if(!success) {
            lcd_write(LCD1, (unsigned char*)syncFailureHdr);
            waitForDelay();
            //strout(LCD2, <<ID NUM>>);
            return;
          }
          break;
        }
    }
    }
    to_tx_mode(sendSyncID, 17);
    to_rx_mode();
    while(one) {
      if(KEYPAD_VALID) {
        key = checkKeypadInput();
        if(key == 'B') {
          syncList[bandID][0] = 0x00;
          rx_reset();
          lcd_write(LCD1, "Sync cancelled.");
          waitForDelay();
          return;
        }
      }
      if(XCVR_IRQ == 0)
    {
        send_read_address(0x7F);
        for (i=0 ; i<17 ; i++)
            RF_RXBUF[i] = send_command();

        if(RF_RXBUF[0] == 0x25 && syncList[bandID][0] == RF_RXBUF[1] && syncList[bandID][1] ==
        RF_RXBUF[2] && syncList[bandID][3] == RF_RXBUF[3])
        {
          lcd_write(LCD1, (unsigned char*)syncSuccessHdr);
          strout(LCD2, (unsigned char*)syncList[bandID]);
          rx_reset();
          break;
        }
    }
    }



    //temporary:
    waitForDelay();

}

void locateWristbands()
{
    lcd_write(LCD1, (unsigned char *)locateHdr);
```



```c
    strout(LCD2, (unsigned char *)locateStop);
    //WIRELESS CODE GOES HERE -> Send beep command

    to_tx_mode(locate, 17);

    waitForEnter();
    to_tx_mode(doneLocate, 17);
    //WIRELESS CODE -> Send stop beep command
}

void setPassword() {
  //unsigned char display[2] = "\00";
  unsigned char data;
  unsigned char i;
  int lcdAddr = 0x41;
  unsigned char confirm[4];
  unsigned char newpwd[4];
  unsigned char valid = 0;
  unsigned char done = 0;
  int j;


  while(!valid)
  {
    lcd_write(LCD1, (unsigned char *)setPwdHdr);
    strout(LCD2, (unsigned char *)pwdPrompt);
    i = 0x04;
    done = 0;
    lcdAddr = 0x41;
      while(!done) { //Enter 4 digit password.
        if(KEYPAD_VALID)
        {
          data = checkKeypadInput();
          if(data == 'B' && state != 0) {
            return;
          }
          if(data == 'C' && i < 4) //backspace pressed
          {
              //display[0] = ' ';
              lcdAddr--;
              strout(lcdAddr, " ");
              i++;
          }
          else if(i > 0 && data != '#' && data != '*' && data != 'A' && data != 'B' && data !=
          'C' && data != 'D')
          {
              newpwd[4-i] = data;
              //display[0] = data;
              strout(lcdAddr, "*");
              lcdAddr++;
              i--;
          }
          else if(data == 'D' && i == 0) //Must press enter key to continue
```



```c
          {
              done = 1;
          }
        }
      }

    lcd_write(LCD1, (unsigned char *)reenterPwdHdr);
    strout(LCD2, (unsigned char *)pwdPrompt);
    lcdAddr = 0x41;
    i = 0x04;
    done = 0;
    while(!done) { //Re-enter password for confirmation
            if(KEYPAD_VALID) {
              data = checkKeypadInput();
              if(data == 'B' && state != 0) {
               return;
            }
              if(data == 'C' && i < 4) //backspace pressed
              {
                //display[0] = ' ';
                lcdAddr--;
                strout(lcdAddr, " ");
                i++;
              }
              else if(i > 0 && data != '#' && data != '*' && data != 'A' && data != 'B' && data
              != 'C' && data != 'D')
              {
                confirm[4-i] = data;
                //display[0] = data;
                strout(lcdAddr, "*");
                lcdAddr++;
                i--;
              }
              else if(data == 'D' && i == 0) //Must press enter key to continue
              {
                  done = 1;
              }
            }
    }
    valid = 1;
    for(j = 0; j < 4; j++) //check two passwords to make sure they match
    {
        if(newpwd[j] != confirm[j])
        {
            valid = 0;
            lcd_write(LCD1, (unsigned char *)pwdMismatch);
            strout(LCD2, (unsigned char *)tryAgain);
            waitForDelay();
            break;
        }
    }

  }
```

```c
  for(j = 0; j < 4; j++) {
    pwd[j] = newpwd[j];
  }
  lcd_write(LCD1, (unsigned char *)pwdSet);
  strout(LCD2, (unsigned char *)pwdChanged2);
  waitForDelay();
}

void waitForDelay()
{
    unsigned char data;
    int counter = 0;
    while(counter < 3000) //Wait for 3 sec or until user presses Enter
    {
        if(KEYPAD_VALID){
            data = checkKeypadInput();
            if(data == 'D')
            {
                return;
            }
        }
        del1m(1);
        counter++;
    }
}

void waitForEnter()
{
    unsigned char data;
    while(one) //Wait for user to press Enter
    {
        if(KEYPAD_VALID){
            data = checkKeypadInput();
            if(data == 'D')
            {
                return;
            }
        }
    }
}

unsigned char enterPwd()
{
    unsigned char data;
    unsigned char i = 0x04;
    int lcdAddr = 0x41;
    unsigned char temppwd[4];
    unsigned char valid = 0;
    unsigned char done = 0;
    int j;
    lcd_write(LCD1, (unsigned char *)enterPwdHdr);
    strout(LCD2, (unsigned char *)pwdPrompt);
    while(!done) { //Enter 4 digit password.
```



```c
        if(KEYPAD_VALID) {
          data = checkKeypadInput();
          if(data == 'B') //Pressed clear
          {
            return 0;
          }
          if(data == 'C' && i < 4) //backspace pressed
          {
            lcdAddr--;
            strout(lcdAddr, " ");
            i++;
          }
          else if(i > 0 && data != '#' && data != '*' && data != 'A' && data != 'B' && data !=
          'C' && data != 'D')
          {
            temppwd[4-i] = data;
            strout(lcdAddr, "*");
            lcdAddr++;
            i--;
          }
          else if(data == 'D' && i == 0) //Must press enter key to continue
          {
            done = 1;
          }
        }
    }
    valid = 1;
    for(j = 0; j < 4; j++) //check two passwords to make sure they match
    {
        if(pwd[j] != temppwd[j])
        {
            valid = 0;
            break;
        }
    }
    if(!valid)
    {
        lcd_write(LCD1, (unsigned char *)incorrectPwd);
        strout(LCD2, (unsigned char *)tryAgain);
        waitForDelay();
        return 0;
    }
    return 1;
}

unsigned char checkKeypadInput() {
    unsigned char r;
    switch(Keypad) {
      case 0b0000:
         r = '1';
         break;
      case 0b0001:
         r = '2';
```



```c
         break;
      case 0b0010:
         r = '3';
         break;
      case 0b0011:
         r = 'A';
         break;
      case 0b0100:
         r = '4';
         break;
      case 0b0101:
         r = '5';
         break;
      case 0b0110:
         r = '6';
         break;
      case 0b0111:
         r = 'B';
         break;
      case 0b1000:
         r = '7';
         break;
      case 0b1001:
         r = '8';
         break;
      case 0b1010:
         r = '9';
         break;
      case 0b1011:
         r = 'C';
         break;
      case 0b1100:
         r = '*';
         break;
      case 0b1101:
         r = '0';
         break;
      case 0b1110:
         r = '#';
         break;
      case 0b1111:
         r = 'D';
         break;
    }
    del1m(70);
    return r;
}


void clearLCD() {
    cmdout(0x01);
}
```



```c
/*
  strout - Print the contents of the character string "s" starting at LCD
  RAM location "x".  The string must be terminated by a zero byte.
*/
void strout(int x, unsigned char *s)
{
    unsigned char ch;

    cmdout(x | 0x80);   // Make A contain a Set Display Address command


    while ((ch = *s++) != (unsigned char) NULL) {
        datout(ch);     // Output the next character
    }
}

/*
  datout - Output a byte to the LCD display data register (the display)
  and wait for the busy flag to reset.
*/
void datout(unsigned char x)
{
    PTB_PTB4 = 1;            // RS=1
    PTB_PTB5 = 0;            // Set R/W=0,
    PTB_PTB6 = 0;            // E=0
    nibout(x);          // Put data bits 4-7 in PTB
    nibout(x << 4);     // Put data bits 0-3 in PTB
    busywt();           // Wait for BUSY flag to reset
}

/*
  cmdout - Output a byte to the LCD display instruction register and
  wait for the busy flag to reset.
*/
void cmdout(unsigned char x)
{
    PTB_PTB4 = 0;            // RS=0
    PTB_PTB5 = 0;            // Set R/W=0,
    PTB_PTB6 = 0;            // E=0
    nibout(x);          // Output command bits 4-7
    nibout(x << 4);     // Output command bits 0-3
    busywt();           // Wait for BUSY flag to reset
}

/*
  nibout - Puts bits 4-7 from x into the four bits of Port B that we're
  using to talk to the LCD.  The other bits of Port B are unchanged.
  Toggle the E control line low-high-low.
*/
void nibout(unsigned char x)
{

    PTB = ((x >> 4) & 0x0f) | (PTB & 0xf0);   // Use bits 0-3
```

```
    PTB_PTB6 = 1;       // Set E to 1
    PTB_PTB6 = 0;       // Set E to 0
}

/*
  initialize - Do various things to force a initialization of the LCD
  display by instructions, and then set up the display parameters and
  turn the display on.
*/
void initialize()
{
    del1m(15);          // Delay at least 15ms

    iniout(0x30);       // Send a 0x30
    del1m(4);           // Delay at least 4msec

    iniout(0x30);       // Send a 0x30
    del40u(3);          // Delay at least 100usec

    iniout(0x30);       // Send a 0x30
    del40u(3);          // Delay at least 100usec

    iniout(0x20);       // Function Set: 4-bit interface
    busywt();           // Wait for BUSY flag to reset

    cmdout(0x28);       // Function Set: 4-bit interface, 2 lines

    cmdout(0x0C);       // Display and cursor on
}

/*
  iniout - Output a byte to the LCD control register.  Same as the "cmdout"
  function but it doesn't wait for the BUSY flag since that flag isn't
  working during initialization.
*/
void iniout(unsigned char x)
{
    PTB_PTB4 = 0;            // RS=0
    PTB_PTB5 = 0;            // Set R/W=0,
    PTB_PTB6 = 0;            // E=0
    nibout(x);
}

/*
  busywt - Wait for the BUSY flag to reset
*/
void busywt()
{
    unsigned char bf;

    DDRB = 0x70;           // Set Data lines for input
    PTB_PTB4 = 0;            // RS=0
```



```
    PTB_PTB5 = 1;            // Set R/W=1,
    PTB_PTB6 = 0;            // E=0
    do {
        PTB_PTB6 = 1;   // Set E=1

        bf = PTB & 0x08;  // Read status register bits 0-3

        PTB_PTB6 = 0;   // Set E=0
        PTB_PTB6 = 1;   // Set E=1, ignore bits 0-3
        PTB_PTB6 = 0;   // Set E=0
    } while (bf != 0);  // If Busy (PTB7=1), loop

    DDRB = 0xff;        // Set PTB for output
}

/*
  Note: these delay routines only work if the the delay0 and delay1
  variables are located in the RAM zero page
*/

/*
  del40u - Delay about 40usec times the "d" argument
*/
void del40u(int d)
{
    while (d--) {
        asm {
            ; The following code delays about 40 microseconds by looping.
            ; Total time is 4 + 13 * (4 + 3) = 95 cycles
            ; A 9.8304MHz external clock gives an internal CPU clock of
            ; 2.4576MHz (407ns/cycle).  95 x .407 usec = 38.7 usec.
            ; The JSR and RTS instructions will add a bit more.
                    mov     #13,delay0  ; 4 cycles
            u1:     dec     delay0      ; 4 cycles
                    bne     u1          ; 3 cycles
        }
    }
}

/*
  del1m - Delay about 1msec times the "d" argument.
*/
void del1m(int d)
{
    while (d--) {
        asm {
            ; The following code delays 1 millisecond by looping.
            ; Total time is 4 + 5 * (4 + 68 * (4 + 3) + 4 + 3) = 2439 cycles
            ; A 9.8304MHz external clock gives an internal CPU clock of
            ; 2.4576MHz (407ns/cycle).  Delay is then .993 milliseconds.
                    mov     #5,delay1   ; 4 cycles
            m1:     mov     #68,delay0  ; 4 cycles
            m0:     dec     delay0      ; 4 cycles
```



```
                    bne     m0          ; 3 cycles
                    dec     delay1      ; 4 cycles
                    bne     m1          ; 3 cycles
        }
    }
}

void write0() {
  XCVR_SCK = 0;
  XCVR_SDI = 0;
  XCVR_SCK = 1;
}

void write1() {
  XCVR_SCK = 0;
  XCVR_SDI = 1;
  XCVR_SCK = 1;
}

void writeByte(char address, char data)
{
  char cnt;
  address |= 0x80;
  XCVR_SCK = 0;
  XCVR_CSN = 0;
  for(cnt=0 ; cnt<8 ; cnt++) {
    if(address & 0x80)
      write1();
    else
      write0();
    address <<= 1;
  }
  for(cnt=0 ; cnt<8 ; cnt++) {
    if(data & 0x80)
      write1();
    else
      write0();
    data <<= 1;
  }
  XCVR_SCK = 0;
  XCVR_CSN = 1;
}

char readByte(char address)
{
  char cnt, read_data;
  address &= 0x7F;
  XCVR_SCK = 0;
  XCVR_CSN = 0;
  for(cnt=0 ; cnt<8 ; cnt++) {
    if(address & 0x80)
      write1();
    else
```



```
      write0();
    address <<= 1;
  }
  read_data = 0x00;
  for(cnt=0 ; cnt<8 ; cnt++) {
    read_data <<= 1;
    XCVR_SCK = 0;
    if(XCVR_SDO == 1)
      read_data |= 0x01;
    XCVR_SCK = 1;
  }
  XCVR_SCK = 0;
  XCVR_CSN = 1;
  return read_data;
}

// Initialize the RFM22 for transmitting
void init_RFM22(void)
{
    writeByte(0x06, 0x00);    // Disable all interrupts
    writeByte(0x07, 0x01);      // Set READY mode
    writeByte(0x09, 0x7F);      // Cap = 12.5pF
    writeByte(0x0A, 0x2B);      // Clk output is 4MHz, on GPIO2

    writeByte(0x0B, 0xF4);      // GPIO0 is for RX data output
    writeByte(0x0C, 0xEF);      // GPIO1 is TX/RX data CLK output
    writeByte(0x0D, 0x00);      // GPIO2 for MCLK output
    writeByte(0x0E, 0x00);      // GPIO port use default value

    writeByte(0x0F, 0x70);      // NO ADC used
    writeByte(0x10, 0x00);      // no ADC used
    writeByte(0x12, 0x00);      // No temp sensor used
    writeByte(0x13, 0x00);      // no temp sensor used

    writeByte(0x70, 0x20);      // No manchester code, no data whiting, data rate < 30Kbps

    writeByte(0x1C, 0x1D);      // IF filter bandwidth
    writeByte(0x1D, 0x40);      // AFC Loop
    //writeByte(0x1E, 0x0A);    // AFC timing

    writeByte(0x20, 0xA1);      // clock recovery
    writeByte(0x21, 0x20);      // clock recovery
    writeByte(0x22, 0x4E);      // clock recovery
    writeByte(0x23, 0xA5);      // clock recovery
    writeByte(0x24, 0x00);      // clock recovery timing
    writeByte(0x25, 0x0A);      // clock recovery timing

    //writeByte(0x2A, 0x18);
    writeByte(0x2C, 0x00);
    writeByte(0x2D, 0x00);
    writeByte(0x2E, 0x00);

    writeByte(0x6E, 0x27);      // TX data rate 1
```

```
    writeByte(0x6F, 0x52);      // TX data rate 0

    writeByte(0x30, 0x8C);      // Data access control

    writeByte(0x32, 0xFF);      // Header control

    writeByte(0x33, 0x42);      // Header 3, 2, 1, 0 used for head length, fixed packet length,
    synchronize word length 3, 2,

    writeByte(0x34, 64);        // 64 nibble = 32 byte preamble
    writeByte(0x35, 0x20);      // 0x35 need to detect 20bit preamble
    writeByte(0x36, 0x2D);      // synchronize word
    writeByte(0x37, 0xD4);
    writeByte(0x38, 0x00);
    writeByte(0x39, 0x00);
    writeByte(0x3A, 's');       // set tx header 3
    writeByte(0x3B, 'o');       // set tx header 2
    writeByte(0x3C, 'n');       // set tx header 1
    writeByte(0x3D, 'g');       // set tx header 0
    writeByte(0x3E, 17);        // set packet length to 17 bytes

    writeByte(0x3F, 's');       // set rx header
    writeByte(0x40, 'o');
    writeByte(0x41, 'n');
    writeByte(0x42, 'g');
    writeByte(0x43, 0xFF);      // check all bits
    writeByte(0x44, 0xFF);      // Check all bits
    writeByte(0x45, 0xFF);      // check all bits
    writeByte(0x46, 0xFF);      // Check all bits

    writeByte(0x56, 0x01);

    writeByte(0x6D, 0x07);      // Tx power to max

    writeByte(0x79, 0x00);      // no frequency hopping
    writeByte(0x7A, 0x00);      // no frequency hopping

    writeByte(0x71, 0x22);      // GFSK, fd[8]=0, no invert for TX/RX data, FIFO mode,
    txclk-->gpio

    writeByte(0x72, 0x48);      // Frequency deviation setting to 45K=72*625

    writeByte(0x73, 0x00);      // No frequency offset
    writeByte(0x74, 0x00);      // No frequency offset

    writeByte(0x75, 0x53);      // frequency set to 434MHz
    writeByte(0x76, 0x64);      // frequency set to 434MHz
    writeByte(0x77, 0x00);      // frequency set to 434Mhz

    writeByte(0x5A, 0x7F);
    writeByte(0x59, 0x40);
    writeByte(0x58, 0x80);
```



```
    writeByte(0x6A, 0x0B);
    writeByte(0x68, 0x04);
    writeByte(0x1F, 0x03);
}

void to_tx_mode(unsigned char buf[], char size)
{
    uchar i;
    writeByte(0x07, 0x01);  // To ready mode
    del1m(50);
    XCVR_RxANT = 0;
    XCVR_TxANT = 1;

    writeByte(0x08, 0x03);  // FIFO reset
    writeByte(0x08, 0x00);  // Clear FIFO
    writeByte(0x34, 64);    // preamble = 64nibble
    writeByte(0x3E, size);  // packet length = 17bytes
    for (i=0; i<size; i++)
    {
        writeByte(0x7F, buf[i]);    // send payload to the FIFO
    }

    writeByte(0x05, 0x04);  // enable packet sent interrupt
    i = readByte(0x03);     // Read Interrupt status1 register
    i = readByte(0x04);

    writeByte(0x07, 9); // Start TX

    while(XCVR_IRQ);
    writeByte(0x07, 0x01);  // to ready mode

    XCVR_RxANT = 0;
    XCVR_TxANT = 0;
}

void rx_reset(void) {
  int i;
  writeByte(0x07, 0x01);
    i = readByte(0x03);     //read Interrupt Status1 reg
    i = readByte(0x04);
    writeByte(0x7E, 0x17);
    writeByte(0x08, 0x03);  //fifo reset
    writeByte(0x08, 0x00);
    writeByte(0x07, 0x05);
    writeByte(0x05, 0x02);
}

void to_rx_mode(void) {
    writeByte(0x07, 0x01);  //to ready mode
    XCVR_RxANT = 1;
    XCVR_TxANT = 0;
    rx_reset();
}
```



```
uchar send_command(void) {
  uchar Result, i;

  XCVR_SCK = 0;
  Result = 0;
  for(i=0 ; i<8 ; i++) {
    Result = Result<<1;
    XCVR_SCK = 1;
    //NOP();
    if(XCVR_SDO)
      Result |= 1;

    XCVR_SCK = 0;
    //NOP();
 }
 return(Result);
}

void send_read_address(uchar addr)
{
  uchar cnt = 8;
  addr &= 0x7f; // spi read funciton

  XCVR_CSN = 1;
  XCVR_SCK = 0;
  XCVR_CSN = 0;
  while(cnt--)  // cycle to 8 times
  {
     if(addr & 0x80)// send one bit command
        write1();
     else
        write0();
     addr = addr << 1;
  }
}

void main(void) {
    unsigned char c[2] = "\00";
      unsigned char key;
      unsigned char pwdCheck = 0;
      unsigned char lcdUpdate = 1;
      int timer = 0;

      uchar i, chksum;

    // EnableInterrupts; /* enable interrupts */
    /* include your code here */

    CONFIG1_COPD = 1;           // disable COP reset


    DDRB = 0xff;                // Set PTB bits 0-7 for output
```



```
    DDRA_DDRA5 = 1;

    DDRD_DDRD0 = 1; //Set bits for RFM22B
    DDRD_DDRD1 = 1;
    DDRD_DDRD2 = 0;
    DDRD_DDRD3 = 1;
    DDRD_DDRD4 = 1;
    DDRD_DDRD6 = 1;
    DDRD_DDRD7 = 0;

    init_RFM22();

    initialize();               // Initialize the LCD display

    //Set cursor back to home

    PTB_PTB4 = 0;            // RS=1
    PTB_PTB5 = 0;            // Set R/W=0,
    PTB_PTB6 = 0;            // E=0
    cmdout(0x02);


    state = 0;
    armed = 0;
      alarmActive = 0;
      ALARM_BUZZER = 1;  //set initial state variables
    setPassword();
    state = 1;

    lcd_write(LCD1, "        SIEMENS         ");    //Show splash screen
    strout(LCD2, "         SIREN          ");
    del1m(5000);

    to_rx_mode(); //start Rx


    while (one) {               // Loop forever
    if(lcdUpdate) {
      lcdUpdate = 0;
        if(state == 1)
        {
            if(armed)
            {
                lcd_write(LCD1, (unsigned char *)menuHdr);
                strout(LCD2, (unsigned char *)menuDisable);
            }
            else
            {
                lcd_write(LCD1, (unsigned char *)menuHdr);
                strout(LCD2, (unsigned char *)menuEnable);
            }
        }
        else if(state == 2)
```

```
        {
            lcd_write(LCD1, (unsigned char *)menuHdr);
            strout(LCD2, (unsigned char *)menuChangePwd);
        }
        else if(state == 3)
        {
            lcd_write(LCD1, (unsigned char *)menuHdr);
            strout(LCD2, (unsigned char *)menuSync);
        }
        else if(state == 4)
        {
            lcd_write(LCD1, (unsigned char *)menuHdr);
            strout(LCD2, (unsigned char *)menuLocate);
        }
        else if(state == 7)
        {
            if(armed)
            {
                lcd_write(LCD1, (unsigned char *)statusHdr);
                strout(LCD2, (unsigned char *)statusEn);
            }
            else
            {
                lcd_write(LCD1, (unsigned char *)statusHdr);
                strout(LCD2, (unsigned char *)statusDis);
            }
        }
    }


    if(KEYPAD_VALID)
    {

      lcdUpdate = 1;
      timer = 0;
        key = checkKeypadInput();
        if(key == '#')
        {
            if(state == 1)// && !armed)
                state = 2;
            else if(state == 2)
                state = 3;
            else if(state == 3)
                state = 4;
            else if(state == 4)
                state = 1;
        }
        else if(key == '*')
        {
            if(state == 1)// && !armed)
                state = 4;
            else if(state == 2)
                state = 1;
```



```
            else if(state == 3)
                state = 2;
            else if(state == 4)
                state = 3;
        }
        else if(key == 'D')
        {
            if(state == 1)
            {
                if(armed)
                {
                    pwdCheck = enterPwd();
                    if(pwdCheck)
                    {
                        disarmSystem();
                        state = 1;
                    }
                }
                else
                {
                    armSystem();
                    state = 7;
                }
            }
            else if(state == 2)
            {
                pwdCheck = enterPwd();
                if(pwdCheck)
                {
                    setPassword();
                }
            }
            else if(state == 3)
            {
                syncWristbands();
            }
            else if(state == 4)
            {
                locateWristbands();
            }
            else if(state == 7)
            {
                state = 1;
            }
        }
        else if(key == 'B')
        {
            if(state == 1 || state == 2 || state == 3 || state == 4)
            {
                state = 7;
            }
        }
        else if(key == 'A')
```



```
        {
            if(state == 7)
                state = 1;
        }
    }
    if(armed) //check for alarm signal
    {

        if(XCVR_IRQ == 0)
        {
            send_read_address(0x7F);
            for (i=0 ; i<17 ; i++)
                RF_RXBUF[i] = send_command();

            if(RF_RXBUF[0] == 0x30) {
              for(i = 0; i < 6; i++) {
                if(RF_RXBUF[1] == syncList[i][0] && RF_RXBUF[2] == syncList[i][1] && RF_RXBUF[3]
                 == syncList[i][2]) {

                  alarmActive = 1;
                    ALARM_BUZZER = 0;
                    lcd_write(LCD1, "ALERT! YOUR KID");
                    strout(LCD2, "IS DROWNING!");
                }
              }
            }

            chksum = 0;
            for(i=0 ; i<17 ; i++)
                chksum += RF_RXBUF[i];
            if(( chksum == RF_RXBUF[16] )&&( RF_RXBUF[0] == 0x30 ))
          writeByte(0x07, 0x01);    //to ready mode
        else
          rx_reset();
        }

        if(alarmActive) {
           ALARM_BUZZER = 0;
           if(state == 7) {
             strout(LCD1, "ALERT! YOUR KID         ");
             strout(LCD2, "IS DROWNING!            ");
           }
        }
    }
    else {
      if(XCVR_IRQ == 0) {
          rx_reset();
      }
    }


    delm(1);
    if(timer > 20000) {
```



```
        lcdUpdate = 1;
        state = 7;
        timer = 0;
      }
      timer++;
    }

}
```

```
/*************************************************************
*       Siemens Re-chargeable Emergency Notifier Wristband (SIREN)
*       EE459, TEAM 5B
*
*       Port A, bit 0 - output to LED anode (1 = LED on, 0 = LED off)
*       Port A, bit 1 - input from water-immersion sensor (0 = H2O, 1 = no H2O)
*       Port A, bit 2 - output to buzzer (1 = off, 0 = on)
*       Port A, bit 3 - input from button (0 = pressed, 1 = not pressed)
*
*       Graphic LCD 84x48 pixels:
*       Port B, bit 0 - LCD chip enable  (active high)
*       Port B, bit 1 - LCD reset (active low)
*       Port B, bit 2 - LCD D/~C (1 = Data being transmitted, 0 = Control)
*       Port B, bit 3 - LCD serial data line
*       Port B, bit 4 - LCD clock for serial interface
*       Port B, bit 5 - LCD backlight power supply
*
*       Transceiver
*       Port D, bit 0 - RxAnt
*       Port D, bit 1 - TxAnt
*       Port D, bit 2 - IRQ
*       Port D, bit 3 - CSN
*       Port D, bit 4 - SCK
*       Port D, bit 6 - SDI
*       Port D, bit 7 - SDO
*
*       On bootup, Siemens logo is shown briefly on LCD then fish sequence runs 5x
*       When the H2O is detected, interrupt is driven and the LED/buzzer comes on
*       When the button is pushed, the LCD turns on and fish sequence runs 5x then turns off
*
*
*************************************************************/

#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

typedef unsigned char uchar;
typedef unsigned int uint;


#define LED_GREEN      PTA_PTA0
#define H2O_SENSE      PTA_PTA1
#define BUZZER         PTA_PTA2
#define BUTTON         !PTA_PTA3
#define LED_RED        PTA_PTA4

#define LCD_SCE        PTB_PTB0
#define LCD_RST        PTB_PTB1
#define LCD_DC         PTB_PTB2
#define LCD_DN         PTB_PTB3
#define LCD_SCLK       PTB_PTB4
#define LCD_BACKLIGHT  PTB_PTB5

#define XCVR_RxANT    PTD_PTD0
```



```
#define XCVR_TxANT    PTD_PTD1
#define XCVR_IRQ      PTD_PTD2
#define XCVR_CSN      PTD_PTD3
#define XCVR_SCK      PTD_PTD4
#define XCVR_SDI      PTD_PTD6
#define XCVR_SDO      PTD_PTD7

#define LCD_CTR  0
#define LCD_DAT  1

#define LCD_WIDTH  84
#define LCD_HEIGHT 48

#define BATTERY_HIGH     3
#define BATTERY_MED      2
#define BATTERY_LOW      1
#define BATTERY_EMPTY    0



void interrupt waterISR(void);
void system_init(void);
void wait500ms(void);
void wait100ms(void);
void wait5ms(void);
void low_battery_detect(void);
void wristband_sync(void);
void locate_wristband(void);

void lcd_init(void);
void lcd_display(unsigned char array_b[], char f);
void lcd_bmp(unsigned char array_b[], unsigned char array_w[], unsigned char array_f[]);
void lcd_full_bmp(unsigned char array[]);
void lcd_sync(unsigned char array_b[]);
void lcd_clear(void);
void lcd_backlight(char enabled);
void lcd_write(char dc, char data);

void send_read_address(uchar addr);
uchar send_command(void);
void to_rx_mode(void);
void to_tx_mode(unsigned char buf[], int size);
void rx_reset(void);
void init_RFM22(void);
char readByte(char address);
void writeByte(char address, char data);
void write1();
void write0();


//Global Variables
uchar water_buf[17] = {0x30, 0xab, 0xcd, 0xef, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B,
0x3C, 0x3D, 0x3E, 0x3F, 0x78};
uchar locate_buf[17] = {0x29, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B,
```



```
0x3C, 0x3D, 0x3E, 0x3F, 0x78};
uchar locate_buf2[17] = {0x28, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B,
 0x3C, 0x3D, 0x3E, 0x3F, 0x78};
uchar sync_send_buf[17] = {0x27, 0xab, 0xcd, 0xef, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A,
0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x78};
uchar sync_rec_buf[17] = {0x26, 0x12, 0x34, 0x56, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B
, 0x3C, 0x3D, 0x3E, 0x3F, 0x78};
uchar sync_confirm_buf[17] = {0x25, 0xab, 0xcd, 0xef, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A,
0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x78};
uchar RF_RXBUF[35];
uchar * batt_level;
uchar base_id[3] = {0x00, 0x00, 0x00};
int battery_level;
uchar one = 1;


#pragma CODE_SEG MY_ISR_ROM

#pragma TRAP_PROC
void dummyISR(void)
{
}

#pragma CODE_SEG DEFAULT


const unsigned char siem [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xF0, 0xF0, 0xF0, 0x70, 0x70, 0x70, 0x70, 0x70, 0x00, 0x00,
0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0x00, 0x00, 0xF0, 0xF0, 0xF0, 0xF0, 0x70, 0x70, 0x70, 0x70, 0x70,
0x00, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0x80, 0x00, 0x00, 0x00, 0x80, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0x00, 0x00, 0x00, 0xF0, 0xF0, 0xF0, 0xF0, 0x70, 0x70, 0x70, 0x70, 0x70, 0x00, 0x00, 0x00,
0xF0, 0xF0, 0xF0, 0xF0, 0x80, 0x00, 0x00, 0xF0, 0xF0, 0xF0, 0xF0, 0x00, 0x80, 0xF0, 0xF0, 0xF0,
0x70, 0x70, 0x70, 0x70, 0x70, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x7F, 0x7F, 0x7F, 0x7F, 0xFC, 0xF0,
0xE0, 0xE0, 0xE0, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x70,
0x70, 0x70, 0x70, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x0F, 0xFF, 0xFF, 0xE0, 0xE0, 0xE0, 0xFF,
0x0F, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x70, 0x70, 0x70, 0x70,
0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x7F, 0x7F, 0xFF, 0xF0, 0x80, 0xFF, 0xFF, 0xFF, 0xFF, 0x00,
0x0F, 0x7F, 0x7F, 0x7F, 0x7F, 0xFC, 0xF0, 0xF0, 0xF0, 0x80, 0x00, 0x00, 0x00, 0x30, 0x30, 0x30,
0x30, 0x30, 0x30, 0x3F, 0x1F, 0x1F, 0x1F, 0x00, 0x3F, 0x3F, 0x3F, 0x3F, 0x3F, 0x00, 0x00, 0x3F,
0x3F, 0x3F, 0x3F, 0x30, 0x30, 0x30, 0x30, 0x30, 0x00, 0x3F, 0x3F, 0x3F, 0x3F, 0x00, 0x00, 0x1F,
0x3F, 0x3F, 0x3F, 0x00, 0x00, 0x3F, 0x3F, 0x3F, 0x3F, 0x00, 0x00, 0x00, 0x3F, 0x3F, 0x3F, 0x3F,
0x30, 0x30, 0x30, 0x30, 0x30, 0x00, 0x00, 0x00, 0x3F, 0x3F, 0x00, 0x00, 0x03, 0x3F, 0x3F, 0x3F,
0x3F, 0x3F, 0x3F, 0x00, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x33, 0x3F, 0x3F, 0x0F, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```



```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const unsigned char siren [] ={
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x46, 0x49, 0x49, 0x49, 0x31,
0x00, 0x00, 0x41, 0x7f, 0x41, 0x00, 0x00, 0x7f, 0x09, 0x19,
0x29, 0x46, 0x00, 0x7f, 0x49, 0x49, 0x49, 0x41, 0x00, 0x7f,
0x04, 0x08, 0x10, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xA0, 0x00, 0xF0, 0x00, 0xFA, 0x00, 0xFF, 0x00,
};

const unsigned char sync [] = {
0x00, 0x00, 0x00, 0x00, 0xC0, 0xE0, 0x20, 0x20, 0x20, 0xE0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x60, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x31, 0x73, 0x42, 0x46, 0x44, 0x7C, 0x38, 0x00,
0x03, 0x0F, 0x3C, 0xF0, 0xF0, 0x3C, 0x0F, 0x03, 0x00, 0x7F, 0x7F, 0x01, 0x01, 0x7F, 0x7E, 0x00,
0x00, 0x3E, 0x7F, 0x41, 0x41, 0x63, 0x22, 0x00, 0x7F, 0x7F, 0x00, 0x00, 0x7F, 0x7F, 0x01, 0x01,
0x7F, 0x7E, 0x00, 0x00, 0x3E, 0x7F, 0x41, 0x41, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x60, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x60, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x60, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x02, 0x02, 0x03, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const unsigned char batt_empty [] = {
0xFF, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0xFF, 0x3C, 0x3C,
};

const unsigned char batt_low [] = {
0xFF, 0x81, 0xBD, 0xBD, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0xFF, 0x3C, 0x3C,
};

const unsigned char batt_med [] = {
0xFF, 0x81, 0xBD, 0xBD, 0x81, 0xBD, 0xBD, 0x81, 0x81, 0x81, 0x81, 0xFF, 0x3C, 0x3C,
};

const unsigned char batt_high [] = {
0xFF, 0x81, 0xBD, 0xBD, 0x81, 0xBD, 0xBD, 0x81, 0xBD, 0xBD, 0x81, 0xFF, 0x3C, 0x3C,
};

const unsigned char fish0 [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0x00, 0x00, 0x70, 0x50, 0x70, 0x00, 0x00, 0x00, 0x07, 0x05, 0x07, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x20, 0x20, 0x40, 0x40, 0xC0, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0E, 0x0A, 0x0E, 0xC0, 0x40,
0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0xFC, 0xFE, 0x7F, 0x1F, 0x07, 0x03,
0x04, 0x88, 0x90, 0x10, 0x20, 0x40, 0x40, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38, 0x28,
0x38, 0x00, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x00, 0x00, 0x20, 0x70, 0x50, 0xD9, 0x06, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```c
void write1()
{
    XCVR_SCK = 0;
    XCVR_SDI = 1;
    XCVR_SCK = 1;
}

void writeByte(char address, char data)
{
    char cnt;
    address |= 0x80;
    XCVR_SCK = 0;
    XCVR_CSN = 0;
    for(cnt=0 ; cnt<8 ; cnt++) {
    if(address & 0x80)
      write1();
    else
      write0();
    address <<= 1;
    }
    for(cnt=0 ; cnt<8 ; cnt++) {
    if(data & 0x80)
      write1();
    else
      write0();
    data <<= 1;
    }
    XCVR_SCK = 0;
    XCVR_CSN = 1;
}

char readByte(char address)
{
    char cnt, read_data;
    address &= 0x7F;
    XCVR_SCK = 0;
    XCVR_CSN = 0;
    for(cnt=0 ; cnt<8 ; cnt++) {
    if(address & 0x80)
      write1();
    else
      write0();
    address <<= 1;
    }
    read_data = 0x00;
    for(cnt=0 ; cnt<8 ; cnt++) {
    read_data <<= 1;
    XCVR_SCK = 0;
    if(XCVR_SDO == 1)
      read_data |= 0x01;
    XCVR_SCK = 1;
    }
    XCVR_SCK = 0;
```



```c
    XCVR_CSN = 1;
    return read_data;
}

// Initialize the RFM22 for transmitting
void init_RFM22(void)
{
    writeByte(0x06, 0x00);      // Disable all interrupts, except LBD interrupt
    writeByte(0x07, 0x41);      // Set READY mode, and enable LBD

    writeByte(0x09, 0x7F);      // Cap = 12.5pF
    writeByte(0x0A, 0x2B);      // Clk output is 4MHz, on GPIO2

    writeByte(0x0B, 0xF4);      // GPIO0 is for RX data output
    writeByte(0x0C, 0xEF);      // GPIO1 is TX/RX data CLK output
    writeByte(0x0D, 0x00);      // GPIO2 for MCLK output
    writeByte(0x0E, 0x00);      // GPIO port use default value

    writeByte(0x0F, 0x70);      // NO ADC used
    writeByte(0x10, 0x00);      // no ADC used
    writeByte(0x12, 0x00);      // No temp sensor used
    writeByte(0x13, 0x00);      // no temp sensor used

    writeByte(0x70, 0x20);      // No manchester code, no data whiting, data rate < 30Kbps

    writeByte(0x1C, 0x1D);      // IF filter bandwidth
    writeByte(0x1D, 0x40);      // AFC Loop
    //writeByte(0x1E, 0x0A);    // AFC timing

    writeByte(0x20, 0xA1);      // clock recovery
    writeByte(0x21, 0x20);      // clock recovery
    writeByte(0x22, 0x4E);      // clock recovery
    writeByte(0x23, 0xA5);      // clock recovery
    writeByte(0x24, 0x00);      // clock recovery timing
    writeByte(0x25, 0x0A);      // clock recovery timing

    //writeByte(0x2A, 0x18);
    writeByte(0x2C, 0x00);
    writeByte(0x2D, 0x00);
    writeByte(0x2E, 0x00);

    writeByte(0x6E, 0x27);      // TX data rate 1
    writeByte(0x6F, 0x52);      // TX data rate 0

    writeByte(0x30, 0x8C);      // Data access control

    writeByte(0x32, 0xFF);      // Header control

    writeByte(0x33, 0x42);      // Header 3, 2, 1, 0 used for head length, fixed packet length,
    synchronize word length 3, 2,

    writeByte(0x34, 64);        // 64 nibble = 32 byte preamble
    writeByte(0x35, 0x20);      // 0x35 need to detect 20bit preamble
    writeByte(0x36, 0x2D);      // synchronize word
```



```c
    writeByte(0x37, 0xD4);
    writeByte(0x38, 0x00);
    writeByte(0x39, 0x00);
    writeByte(0x3A, 's');       // set tx header 3
    writeByte(0x3B, 'o');       // set tx header 2
    writeByte(0x3C, 'n');       // set tx header 1
    writeByte(0x3D, 'g');       // set tx header 0
    writeByte(0x3E, 17);        // set packet length to 17 bytes

    writeByte(0x3F, 's');       // set rx header
    writeByte(0x40, 'o');
    writeByte(0x41, 'n');
    writeByte(0x42, 'g');
    writeByte(0x43, 0xFF);      // check all bits
    writeByte(0x44, 0xFF);      // Check all bits
    writeByte(0x45, 0xFF);      // check all bits
    writeByte(0x46, 0xFF);      // Check all bits

    writeByte(0x56, 0x01);

    writeByte(0x6D, 0x07);      // Tx power to max

    writeByte(0x79, 0x00);      // no frequency hopping
    writeByte(0x7A, 0x00);      // no frequency hopping

    writeByte(0x71, 0x22);      // GFSK, fd[8]=0, no invert for TX/RX data, FIFO mode,
    txclk-->gpio

    writeByte(0x72, 0x48);      // Frequency deviation setting to 45K=72*625

    writeByte(0x73, 0x00);      // No frequency offset
    writeByte(0x74, 0x00);      // No frequency offset

    writeByte(0x75, 0x53);      // frequency set to 434MHz
    writeByte(0x76, 0x64);      // frequency set to 434MHz
    writeByte(0x77, 0x00);      // frequency set to 434Mhz

    writeByte(0x5A, 0x7F);
    writeByte(0x59, 0x40);
    writeByte(0x58, 0x80);

    writeByte(0x6A, 0x0B);
    writeByte(0x68, 0x04);
    writeByte(0x1F, 0x03);
}

void to_tx_mode(unsigned char buf[], int size)
{
    uchar i;
    writeByte(0x07, 0x41);      // Set READY mode, and enable LBD
    XCVR_RxANT = 0;
    XCVR_TxANT = 1;

    writeByte(0x08, 0x03);  // FIFO reset
```



```c
    writeByte(0x08, 0x00);  // Clear FIFO
    writeByte(0x34, 64);    // preamble = 64nibble

    writeByte(0x3E, size);  // packet length = 17bytes
    for (i=0; i<size; i++)
    {
        writeByte(0x7F, buf[i]);    // send payload to the FIFO
    }

    writeByte(0x05, 0x04);  // enable packet sent interrupt
    i = readByte(0x03);     // Read Interrupt status1 register
    i = readByte(0x04);

    writeByte(0x07, 0x49);  // Start TX

    while(XCVR_IRQ);
    writeByte(0x07, 0x41);      // Set READY mode, and enable LBD

    XCVR_RxANT = 0;
    XCVR_TxANT = 0;
    LED_RED = 1;    //red LED blinks on whenever something is transmitted
}

void rx_reset(void)
{
    int i;
    writeByte(0x07, 0x41);  // Set READY mode, and enable LBD
    i = readByte(0x03);     //read Interrupt Status1 reg
    i = readByte(0x04);
    writeByte(0x7E, 0x17);
    writeByte(0x08, 0x03);  //fifo reset
    writeByte(0x08, 0x00);
    writeByte(0x07, 0x45);
    writeByte(0x05, 0x02);
}

void to_rx_mode(void)
{
    writeByte(0x07, 0x41);      // Set READY mode, and enable LBD
    XCVR_RxANT = 1;
    XCVR_TxANT = 0;
    rx_reset();
}

uchar send_command(void)
{
    uchar Result, i;
    XCVR_SCK = 0;
    Result = 0;
    for(i=0 ; i<8 ; i++) {
    Result = Result<<1;
    XCVR_SCK = 1;
    //NOP();
    if(XCVR_SDO)
```

```
      Result |= 1;

    XCVR_SCK = 0;
    //NOP();
    }
    return(Result);
}

void send_read_address(uchar addr)
{
    uchar cnt = 8;
    addr &= 0x7f; // spi read funciton

    XCVR_CSN = 1;
    XCVR_SCK = 0;
    XCVR_CSN = 0;
    while(cnt--)  // cycle to 8 times
    {
     if(addr & 0x80)// send one bit command
        write1();
     else
        write0();
     addr = addr << 1;
    }
}

void low_battery_detect(void)
{
    //0x1A --- Low Battery Detector Threshold 8'bzzzxxxxx
    //0x1B --- Read Battery Voltage Level 8'b000xxxxx
    //set voltage thresholds to operate with voltage capabilities of uC and LCD
    //to deal with rx interrupts also on the IRQ pin, read 0x1B voltage ADC to determine if
    battery level should change
    uchar volt_lvl1, volt_lvl2, volt_lvl3, volt_lvl4, volt_lvl5, voltage_level, i, j, temp;
    uchar volt_array[5];

    volt_lvl1 = readByte(0x1B);
    volt_lvl2 = readByte(0x1B);
    volt_lvl3 = readByte(0x1B);
    volt_lvl4 = readByte(0x1B);
    volt_lvl5 = readByte(0x1B);

    volt_array[0] = volt_lvl1;
    volt_array[1] = volt_lvl2;
    volt_array[2] = volt_lvl3;
    volt_array[3] = volt_lvl4;
    volt_array[4] = volt_lvl5;

  for (j=0 ; j< 5; j++) {
    for (i=0 ; i<4 ; i++) {
      if(volt_array[i] > volt_array[i+1]) {
        temp = volt_array[i+1];
        volt_array[i+1] = volt_array[i];
        volt_array[i] = temp;
```



```
      }
    }
  }
    voltage_level = volt_array[2];  //the median of 5 readings

  if (voltage_level > 0x1C) {       //voltage is > 3.05V (ADC=28)
    batt_level = batt_high;
  }
  else if (voltage_level > 0x17) {  //voltage is > 2.8V (ADC=23)
    batt_level = batt_med;
  }
  else if (voltage_level > 0x13) {  //voltage is > 2.6V (ADC=19)
    batt_level = batt_low;
  }
  else if (voltage_level <= 0x13) { //voltage is <= 2.6V (ADC=19)
    batt_level = batt_empty;
  }
  else {
    BUZZER = 0;
  }
  to_rx_mode();
}

void wristband_sync(void)
{
    uchar i;
    int timer = 0;
    //if SYNC button pressed, transmit SYNC signal to base station (wristband in TX mode)
    to_tx_mode(sync_send_buf, 17);

    //wait for response --- handshaking
    to_rx_mode();
    lcd_clear();
    while(one)
    {
        LED_GREEN = 1;  //green LED goes on when in sync mode, turns off when sync
        complete/failed
        lcd_sync(batt_level); //show animated connectivity bars on LCD

        if(XCVR_IRQ == 0)
        {
            send_read_address(0x7F);
          for (i=0 ; i<17 ; i++)
                RF_RXBUF[i] = send_command();
          if(RF_RXBUF[0] == 0x26)         //received response from base station
          {
              base_id[0] = RF_RXBUF[1]; //saving base station's id into a global variable
              base_id[1] = RF_RXBUF[2];
              base_id[2] = RF_RXBUF[3];
              rx_reset();
              to_tx_mode(sync_confirm_buf, 17); //send confirmation signal to base station that
              we got their id
              LED_GREEN = 0;
              //lcd_full_bmp(happy);
```



```
              lcd_bmp(batt_level, siren, sync_succ); //sync successful screen on LCD
              wait100ms();
              to_rx_mode();
              break;
          }
        }

      timer++;
      if (timer > 7)     //if response is not received in X sexonds
      {
          LED_GREEN = 0;
          //lcd_full_bmp(sad);
          lcd_bmp(batt_level, siren, sync_fail); //sync fail screen on LCD
          wait100ms();
          to_rx_mode();
          break;
      }
    }

}

void locate_wristband (void)
{
    uchar i;
    wait5ms();
    LED_RED = 1;
    send_read_address(0x7F);
        for (i=0 ; i<17 ; i++)
            RF_RXBUF[i] = send_command();
        if(RF_RXBUF[0] == 0x29)   //1st byte of 1st word matches
        {
            LED_GREEN = 1;
            if(RF_RXBUF[1] == base_id[0] && RF_RXBUF[2] == base_id[1] && RF_RXBUF[3] == base_id[
            2])  //next 3 bytes match the base id
            {
                BUZZER = 0;
                rx_reset();
                while(one)
                {
                    if(XCVR_IRQ == 0)
                    {
                        send_read_address(0x7F);
                        for (i=0 ; i<17 ; i++)
                            RF_RXBUF[i] = send_command();
                        if(RF_RXBUF[0]==0x28)  ////1st byte of 2nd word matches
                        {
                            if(RF_RXBUF[1] == base_id[0] && RF_RXBUF[2] == base_id[1] &&
                            RF_RXBUF[3] == base_id[2]) //next 3 bytes match the base id
                            {
                                BUZZER = 1;
                                to_rx_mode();
                                return;
                            }
                        }
```



```
                        }
                        if(BUTTON == 1)
                        {
                            BUZZER = 1;
                            to_rx_mode();
                            return;
                        }
                    }
                }
            }
        else
            to_rx_mode();

}

void lcd_write(char dc, char data)
{
    int i;
    if (dc)
        LCD_DC = 1;
    else
        LCD_DC = 0;
    LCD_SCE = 0;
    for (i=0; i<8; i++)
    {
        if(data & 0x80)
            LCD_DN = 1;
        else
            LCD_DN = 0;
        LCD_SCLK = 0;
        LCD_SCLK = 1;
        data <<=1;
    }
    LCD_SCE = 1;
    LCD_DN = 0;
}

void lcd_clear(void)
{
    int i;
    for (i=0; i<LCD_WIDTH*LCD_HEIGHT/8; i++)
        lcd_write(LCD_DAT, 0x00);
}

void lcd_sync(unsigned char array_b[])
{
  int i;
  for (i=0; i<14; i++)
     lcd_write(LCD_DAT, array_b[i]);
  for (i = 0; i < 70; i++)
        lcd_write(LCD_DAT, siren[i]);
  for (i=0; i <252; i++)
    lcd_write(LCD_DAT, sync[i]);
  for (i=0; i<6; i++)
```

```
    lcd_write(LCD_DAT, 0x00);
  for (i=0; i<8; i++)
    lcd_write(LCD_DAT, 0xFF);
  for (i=0; i<8; i++)
    lcd_write(LCD_DAT, 0x00);
  wait100ms();
  for (i=0; i<8; i++)
    lcd_write(LCD_DAT, 0xFF);
  for (i=0; i<8; i++)
    lcd_write(LCD_DAT, 0x00);
  wait100ms();
  for (i=0; i<8; i++)
    lcd_write(LCD_DAT, 0xFF);
  for (i=0; i<8; i++)
    lcd_write(LCD_DAT, 0x00);
  wait100ms();
  for (i=0; i<8; i++)
    lcd_write(LCD_DAT, 0xFF);
  for (i=0; i<8; i++)
    lcd_write(LCD_DAT, 0x00);
  wait100ms();
  for (i=0; i<8; i++)
    lcd_write(LCD_DAT, 0xFF);
  for (i=0; i<90; i++)
    lcd_write(LCD_DAT, 0x00);
  wait100ms();
  lcd_clear();
}

void lcd_full_bmp(unsigned char array[])
{
    int i;
    for (i = 0; i < LCD_WIDTH*LCD_HEIGHT/8; i++)
        lcd_write(LCD_DAT, array[i]);
}

void lcd_bmp(unsigned char array_b[], unsigned char array_w[], unsigned char array_f[])
{
    int i;
    for (i = 0; i < 14; i++)
        lcd_write(LCD_DAT, array_b[i]);
    for (i = 0; i < 70; i++)
        lcd_write(LCD_DAT, array_w[i]);
    for (i = 0; i < 420; i++)
        lcd_write(LCD_DAT, array_f[i]);
}

void lcd_display(unsigned char array_b[], char f)
{
  wait100ms();
  if(f==0)
      lcd_bmp(array_b, siren, fish0);
  if(f==1)
      lcd_bmp(array_b, siren, fish1);
```



```
  if(f==2)
      lcd_bmp(array_b, siren, fish2);
  if(f==3)
      lcd_bmp(array_b, siren, fish3);
  if(f==4)
      lcd_bmp(array_b, siren, fish4);
}

void lcd_backlight(char enabled)
{
    if (enabled)
        LCD_BACKLIGHT = 1;
    else
        LCD_BACKLIGHT = 0;
}

void lcd_init(void)
{
    LCD_SCE = 1;
    LCD_BACKLIGHT = 1;
    LCD_RST = 0;
    wait100ms();
    LCD_RST = 1;

    lcd_write(LCD_CTR, 0x21); // extended instruction set
    lcd_write(LCD_CTR, 0xA9); // set Vop to 5V
    lcd_write(LCD_CTR, 0x04); // set the temperature coefficient
    lcd_write(LCD_CTR, 0x15); // set the bias system
    lcd_write(LCD_CTR, 0x20); // active mode, horizontal addressing, basic instruction set
    lcd_write(LCD_CTR, 0x0C); // normal mode
    lcd_clear(); // clear the LCD
    lcd_full_bmp(siem);  //show Siemen's logo
    wait500ms();
    lcd_clear;
}

void wait5ms(void)
{
    int i;
    for(i = 0; i<650; i++)
    {
    ;   // delay 5ms
    }
}

void wait100ms(void)
{
    unsigned char j;
    for(j = 0; j<20; j++)  // delay 100ms = 20*5ms
    {
        wait5ms();
    }
}
```



```
void wait500ms(void)
{
    unsigned char j;
    for(j = 0; j<5; j++)  // delay 500ms = 5*100ms
    {
        wait100ms();
    }
}

void system_init(void)
{
    //configure I/O pins
    DDRB = 0x3f;              // LCD pins (port B pins 0-5) as outputs
    DDRD = 0x5B;              // Set PTD bits 0-7 for XCVR in/out signals
    DDRA_DDRA0 = 1;           // Set PTA bit 0 for output (LED)
    PTAPUE_PTAPUE1 = 1;       // Enable pull-up for water switch on PTA1
    DDRA_DDRA2 = 1;           // Set PTA bit 2 for output (buzzer)
    PTAPUE_PTAPUE3 = 1;       // Enable pull-up for button on PTA3
    DDRA_DDRA4 = 1;           // Set PTA bit 4 for output (LED)

    //initialize interrupts
    EnableInterrupts;
    KBSCR_IMASKK = 1;   //mask keyboard interrupts
    KBIER_KBIE1 = 1;    //enable keyboard interrupt on bit 1 (water sensor)
    KBSCR_ACKK = 1;     //disable any false interrupts that may have occurred
    KBSCR_IMASKK = 0;   //clear mask bit for keyboard interrupts

    CONFIG1_COPD =1;    //disable COP reset

    lcd_init();         //initialize the LCD
    init_RFM22();       //initialize all RFM22 registers
    LED_GREEN = 0;
    LED_RED = 0;
    BUZZER = 1;
    writeByte(0x1A, 0x1F);      //set LBD threshold to full battery > 3.2V (ADC=31)
    batt_level = batt_high;
    battery_level = BATTERY_HIGH;
    to_rx_mode();       //default in RX mode
}

void interrupt waterISR(void)  //interrupt service routine for if water is sensed
{
    to_tx_mode(water_buf, 17); //send packet alerting base station that child is drowning
    to_rx_mode();
    KBSCR_ACKK = 1;    //sends ACK to keyboard interrupt to clear the register
}

int main(void)
{
    int timer = 0;
    uchar state = 1;
    uchar i;
    uchar frame = 0;
    BUZZER = 1;
```



```
    system_init();
    wait100ms();


    while(one)
    {
        LED_GREEN = 0;
        LED_RED = 0;
        BUZZER = 1;

        if(XCVR_IRQ == 0)
          locate_wristband();
        low_battery_detect();

        if(state == 0)  //idle state (LCD off)
        {
            if(BUTTON)
            {
                state = 1;
                lcd_backlight(1);
                frame = 0;
                timer = 0;
            }
        }
        else if(state == 1)  //LCD on, fish swims 5x
        {
            if(BUTTON)
            {
                wait100ms();
                if(BUTTON) //check to see that button has been held down
                    state = 2;
            }
            lcd_display(batt_level, frame);
            timer++;
            frame++;
            if(timer > 40)
            {
                state = 0;
                lcd_clear();
                lcd_backlight(0);
            }
            if(frame > 4)
                frame = 0;
        }
        else if(state == 2) //if button held down when screen is on, go into sync mode
        {
          wristband_sync();
          state = 1;
        }
        wait5ms();
    }
}
```

```
/* This is a linker parameter file for the mc68hc908jl16 */

NAMES END /* CodeWarrior will pass all the needed files to the linker by command line. But here
you may add your own files too. */

SEGMENTS /* Here all RAM/ROM areas of the device are listed. Used in PLACEMENT below. */
    Z_RAM                    =  READ_WRITE   0x0060 TO 0x00FF;
    RAM                      =  READ_WRITE   0x0100 TO 0x025F;
    ROM0                     =  READ_ONLY    0xBC00 TO 0xBC0F;
    ROM                      =  READ_ONLY    0xBC10 TO 0xFBFF;
 /* INTVECTS                 =  READ_ONLY    0xFFDE TO 0xFFFF; Reserved for Interrupt Vectors */
END

PLACEMENT /* Here all predefined and user segments are placed into the SEGMENTS defined above. */
    DEFAULT_RAM                        /* non-zero page variables */
                                       INTO  RAM;

    _PRESTART,                         /* startup code */
    STARTUP,                           /* startup data structures */
    ROM_VAR,                           /* constant variables */
    STRINGS,                           /* string literals */
    VIRTUAL_TABLE_SEGMENT,             /* C++ virtual table segment */
    DEFAULT_ROM,
    COPY                               /* copy down information: how to initialize variables */
                                       INTO  ROM;

    MY_ISR_ROM                         INTO  ROM0;

    _DATA_ZEROPAGE,                    /* zero page variables */
    MY_ZEROPAGE                        INTO  Z_RAM;
END


STACKSIZE 0x50

VECTOR 0 _Startup /* Reset vector: this is the default entry point for an application. */
VECTOR ADDRESS 0xFFE0 waterISR
VECTOR ADDRESS 0xFFF6 dummyISR
VECTOR ADDRESS 0xFFF8 dummyISR
VECTOR ADDRESS 0xFFFA dummyISR
```

Signature Sheet

	Team 5A			Team 5B		
	Chris	**Alex**	**Steven**	**Baur**	**Hieu**	**Steph**
System Design	40%	30%	30%	35%	33%	32%
Hardware Design	40%	35%	25%	35%	40%	25%
Hardware Simulation	33%	33%	33%	25%	40%	35%
Hardware Assembly	10%	80%	10%	40%	30%	30%
Hardware Check-out	55%	40%	5%	33%	33%	33%
Microcontroller Software Design	5%	15%	80%	20%	30%	50%
Microcontroller Software Check-out	10%	40%	50%	20%	30%	50%
System Integration	33%	33%	33%	33%	33%	33%
Documentation	31%	35%	34%	20%	60%	20%
Project Report (oral)	56%	22%	22%	33%	33%	33%
Project Report (written)	30%	35%	35%	30%	30%	40%

Christopher Cortes _____

Alexander Nobles _____

Steven Spinn _____

Bauyrzhan Krykpayev _____

Hieu Nguyen _____

Stephanie Shinohara _____