DESIGN AND VERIFICATION OF A SURFACE PLASMON RESONANCE BIOSENSOR

A Thesis
Presented to
The Academic Faculty

By

Daniel R. Sommers

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Bioengineering

Georgia Institute of Technology

September 2003

DESIGN AND VERIFICATION OF A SURFACE PLASMON RESONANCE BIOSENSOR

Approved by:

Dr. William D. Hunt, Advisor	Dr. Allen M. Orville
Dr. Cheng Zhu	Mr. Doug Armstrong
Dr. Larry A. Bottomley	Date Approved Sept 9, 2003

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

BAW – Bulk Acoustic Wave	

BAW – Bulk Acoustic Wave

CCD – Charge Coupled Device

DES – Data Encryption Standard

DSP – Digital Signal Processor

FIPS – Federal Information Processing Standards Publications

FITC – fluorescein isothiocyanate

IEEE - the Institute of Electrical and Electronics Engineers, Inc.

IR – Infrared Light

LR – Linear Regulator

MHz – MegaHertz (1x10^{6} cycles per second)

ms – millisecond (1x10^{-3} seconds)

NIST – National Institute of Standards and Technology

OO – Object Oriented Software

QCM – Quart Crystal Microbalance

SAW – Surface Acoustic Wave

SPR – Surface Plasmon Resonance

TAE – Tris Base Acedic Acid EDTA

UI – User Interface

us – microsecond (1x10^{-6} seconds)

VU – Ultra Violet Light

SUMMARY

The Microelectronics Group has been researching sensors useful for detecting and quantifying events in biological molecular chemistry, for example, binding events. Our previous research has been based primarily on quartz resonators. This thesis describes the results of our initial research of Surface Plasmon Resonance (SPR) based technology. This study contains the design and implementation of a fully functional SPR biosensor with detailed disclosure of monolayer construction, digital hardware interfaces and software algorithms for process the SPR sensor's output.

An antibody monolayer was constructed on the biosensor surface with the goal of setting the strengths, weaknesses and limitation of measuring molecular events with SPR technology. We documented several characteristics of molecular chemistry that directly effect any measurements made using Surface Plasmon Resonance technology including pH, free ions, viscosity and temperature. Furthermore, the component used in our study introduced additional limitations due to wide variations amongst parts, the constraint of a liquid medium and the large surface area used for molecular interrogation. We have identified viable applications for this sensor by either eliminating or compensating for the factors that affect the measured results.

This research has been published at the inaugural IEEE sensors conference and to our knowledge is the first time a biosensor has been constructed by attaching a sensor to a PDA and performing all signal processing, waveform analysis and display in the PDA's core processor.

CHAPTER 1: INTRODUCTION

This Thesis presents the conception, design, implementation and verification of a fully functional biosensor based on Surface Plasmon Resonance (SPR) technology. As shown in Figure 1, the biosensor consists of a Texas Instruments Spreeta^{TM} Surface Plasmon Resonance device [1], interfaced to a Motorola Accompli 009^{TM} Wireless GSM graphical phone [2] via simple digital logic (e.g. no DSP or signal processing capability in the digital interface). We wrote the software (see Appendix B "Software Listing") to read the data from the sensor, build and interpret the waveforms, drive the display, and for broadcast the wireless notification messages on Motorola's FlexOS^{TM} real-time operating system in "C" and Motorola's FlexScript^{TM} Application language.

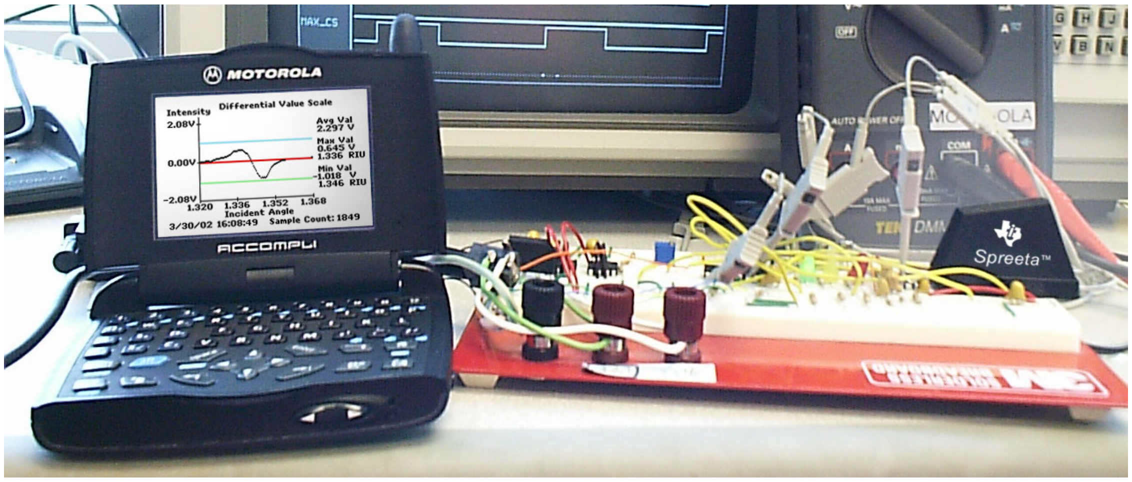


Figure 1. Capture of Biosensor Hardware and User Interface

The monolayer was based on antibody-analyte interactions [3] as illustrated in Figure 2. The bulk of the effort to characterize the biosensor design was to build the monolayer on the gold surface as described in detail later in this document, and attempt to detect the binding events by exposing both positive and negative analytes to the biolayer with the expectation that there will be a predictable, measurable and repeatable signature difference between "true" and "false" analytes.

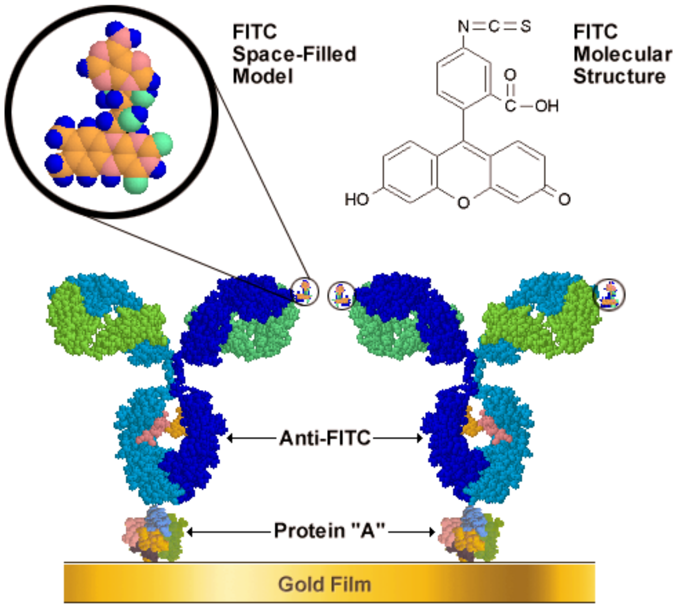


Figure 2. Illustration of Anti-FITC Antibody Immobilized on Gold Surface by Protein "A" with tightly bound FITC Analyte

1.1 GOAL

Surface Plasmon Resonance Technology has been well established as a useful analytical tool for biosensor applications. Among other strengths, it offers a biomaterial interrogation method that doesn't (theoretically) disturb the biomaterial.

This thesis began with an industry survey of the available biosensor technology, and the discovery that Texas Instruments offers a SPR component with sensor surface, biological to electrical transducer and analog electrical interface all contained in a surprisingly small package (~1 in. x 1 in. x ½ in.). This SPR biosensor has many other appealing features such as: portable, low cost (~$35/ea), low power and readily available. Based on this sensor component, our portable self-contained biosensor system was implemented using the following steps:

a) Build a portable hardware interface to this biosensor

b) Write software to pull the data out of the biosensor

c) Write software to interpret and display the data

d) Apply the biosensor to running antibody-analyte assays

The goal was to learn the strengths and weaknesses of SPR as a technology for a biosensor in a continuous monitoring application such as cocaine detection in an airport [5],[7]. And a more specific goal was to learn the strengths and weaknesses of this Spreeta device. Thus this thesis will overview the technology of SPR, document the hardware and software implemented in building the biosensor system and document the results of the assays and the possible sources of error. The conclusion will then recommend possible applications for industry and/or academic research for this biosensor system based on its capabilities and limitations.

1.2 MOTIVATION

Recently, the industry has been calling for “substance detectors” in applications ranging from protecting solders (and citizens) during biological warfare [8],[9] to preventing contamination or spoilage in food preparation and transport [10]. The common principle underlying these types of applications is that of detecting a shift in their environment from “normal” to “dangerous”. Conveniently, the definitions of normal and dangerous are easily characterized and quantified so the challenge is to design a biosensor that can accurately measure and report changes in diverse environments.

One of the characteristics of a good research center is diversity. In that regard the microelectronics and acoustics research center at Georgia Institute of Technology is no exception. This group has performed extensive research on mass changes in crystal technologies (Quartz Crystal Microbalance, Bulk Acoustic Wave, Surface Acoustic Wave, etc.), which result in changes in the resonant frequency of the crystal. One technology previously unexploited at the lab was Surface Plasmon Resonance (SPR). The research strategy for this group is to implement practical biosensor solutions based on demand from industrial and/or research applications.

1.3 SYSTEM OPERAION OVERVIEW

After verifying the requisite components are all readily available from industry standard sources, the basic operational capabilities of the system and defined the theoretical deployment of the system were proposed as follows:

1) ***SPR Theory of Operation****:* the underlying principle behind SPR is that a thin layer of gold will act as a mirror and reflect light projected at it except under special "matching" conditions. At the resonant frequency the light energy will not be reflected, but rather converted into a collective oscillation of electrons in the gold film, generating the plasmon waveform. Thus by varying the frequency of the light striking the back of the gold film, one particular wavelength will resonate with the aggregate permittivity on the front side. A direct plot of light frequency vs. reflected light intensity will show a marked loss of reflected energy at the matching (resonant) wavelength
2) ***Antibody Monolayer Theory of Operation:*** the goal is to completely cover the gold SPR film with a consistent, continuous sheet of antibodies which have been grown to specifically bind only to the antigen of interest (Anthrax, cocaine, etc.). When the analyte encounters the antibody, it will "stick" and thereby cause

a perturbation in the permittivity, which will be measured with the SPR technology.

3) ***Graphical User Interface:*** the main output of an SPR system is a 2-dimensional graph of light wavelength vs. absorbed light. The system must be sensitive enough to measure shifts in the minimum value from one wavelength to another. Further, the user must be able to graphically set thresholds for the maximum shift, enter e-mail addresses for notification, and calibrate the average value output.

4) ***Continuous Operation:*** the system must monitor its environment continuously but conserve battery life. This requires two modes: a relatively slow “wake and sniff” mode and upon detect of an interesting level switch to a “constant monitor” mode. Thus the detection threshold must include a user-definable level that places the system on “alert status”.

5) ***Autonomous Action:*** upon detection of a violation of the acceptable limits set during initialization, the system must generate a response and in this case, a wireless notification to any number of programmable receipts. If this alert is on a public network (e.g. cellular, paging, etc.) then the notification must be secured such that it can be neither lost nor spoofed. The notification can be sent to a human, or to a server, which is programmed to take specific action upon receipt of a detection message.

CHAPTER 2: SYSTEM DESIGN

Although the overall goal of this research was to demonstrate and evaluate the relative strengths and weaknesses of Surface Plasmon Resonance technology, a secondary goal was to build a viable biosensor that could satisfy actual use cases in the industry.

2.1 DESIGN GOALS

Based on an industry survey of electronic molecular detection devices [5] and hand-held wireless devices in general [2], the following list of system requirements was generated:

- Low Cost - $100's instead of $1,000's
- Rugged - no delicate calibration
- Portable - battery operated, hand-held
- Wireless - notification upon detection
- Autonomous - continuous, unmanned monitoring

2.2 MAJOR COMPONENT SELECTION

We met the design goals for building the sensor using industry standard parts. The SPR component was the Texas Instruments Spreeta; the antibody component was mouse monoclonal anti-fluorescein isothiocyanate (anti-FITC) antibodies from Sigma Chemical Company and the PDA was the Motorola Accompli 009.

2.2.1 TEXAS INSTURMENTS SPREETA

The biosensor component that meets the system requirements is the Spreeta [3,4]. It is an industry-standard component that is readily available for bulk purchase, thereby satisfying the low cost goal. The battery in a wireless device meets the Spreeta's power demands thereby satisfying the portability criterion. The Spreeta is also satisfactorily rugged since it requires no delicate calibration, has no moving parts and imposes no constraint on orientation relative to gravity.

The Spreeta biosensor is the only biosensor device that is commercially available that satisfies all of the design constraints:

a) Portable (light weight and easily hand-held)
b) Low power (power requirements serviceable by a battery)
c) Low cost (manufactured using high-volume processes)

2.2.2 SIGMA CHEMICAL COMPANY ANTI-FITC

The selection of the biological component was driven by several factors, not the least of which is the fact that research on the quartz-based sensors had been based on anti-FITC [6] and not only was that component well characterized, it was also already in stock. The fact that the Spreeta sensor supports only liquid assay drove the selection of the analyte: for best results the analyte should be easily dissolved into a liquid medium. The actual analyte, FITC, is very hydrophobic and tends to settle out of solution rapidly whereas urinine is more hydrophilic. The binding affinity of anti-FITC for FITC is comparable to its binding affinity for urinine.

Finally, the negative testing component for anti-FITC binding is Alexa Fluor®, which has the same basic structure as FITC and uranine and therefore the same mass, but has charged ions that prevent it from binding with an antibody raised specifically for FITC. Note that all of these analytes fluoresce in UV light, thereby allowing for cross-correlation of their binding at the sensor surface with alternative measurement schemes like direct light observation through a confocal microscope.

In summary, the biolayer component selections were driven by:

a) pre-characterized monolayer assembly process
b) well known binding affinities for both positive and negative assay
c) cross-correlation techniques for sensor characterization

2.2.3 MOTOROLA ACCOMPLI 009

An understanding of the Spreeta device leads to a system design that can accommodate the constraints. A system that meets the above constraints must be even smaller and less expensive than a lap top computer, so the choice was Motorola's Accompli 009 (A009) wireless phone [2] as the biosensor's main controller. It is fully programmable, has a large color display, and an expansion port to use for biosensor attachment. Additionally, it expands the "portable" concept to include "remote" because the wireless technology has the capability to notify upon detection.

Thus the choice to host the software and interface to the Spreeta was the Motorola Accompli 009 because it satisfied the design goals with the following strengths:

a) It has a real-time operating system
b) It has a synchronous serial port (SPI) hardware driver
c) A full development environment is available for the device
d) The final design will be hand-held (as opposed to a SPI card in a PC)

CHAPTER 3: SPR THEORY OF OPERATION

Surface Plasmon Resonance [11] is a physical phenomenon for which the electrons in a thin gold film can be excited into a collective oscillation by bombarding it with photons (light) of a particular frequency. This phenomenon is analogous to resonant frequency in a mechanical system in that there must be a precise match between the light's frequency and momentum and the characteristics of the gold. Any mismatch between the wavelength of the light: either too short (a.k.a. "blue-shifted") or too long (a.k.a. "red-shifted"), and the gold will fail to exhibit the plasmon oscillation. The matching wavelength is also dependant on the permittivity of the material on the opposite side of the gold film (see Figure 3).

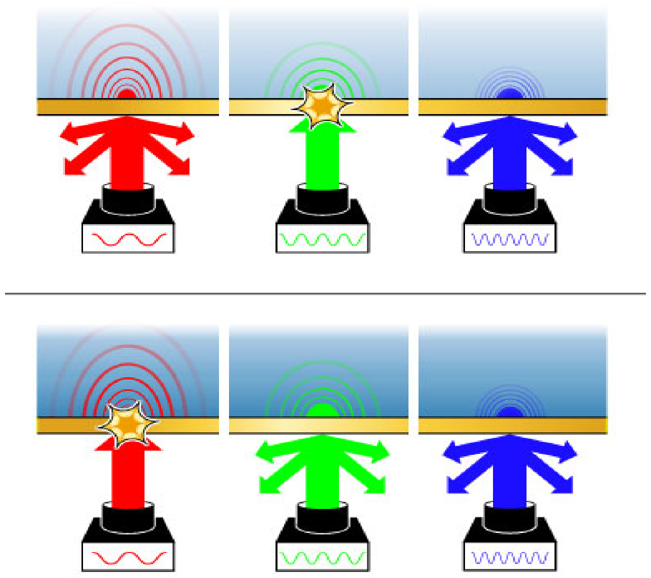


Figure 3. Change in permittivity shifts SPR excitation

Permittivity is a measure of a material's capability to store energy in an electric field and is defined in terms of the material's dielectric constant.

3.1 THE “IDEAL” SPR SYSTEM

In a thin film, the light actually induces an interrogation field on the opposite side of the gold. This field interrogates the relative permittivity for a distance on the order of 1 wavelength of the light striking the surface. A biological system is not likely to be disturbed by an electromagnetic field of low optical intensity. A biological system can, however, invoke a change in permittivity. This change of permittivity within the interrogation field of the SPR system can cause a measurable shift in the wavelength required to excite the plasma state (see Figure 4).

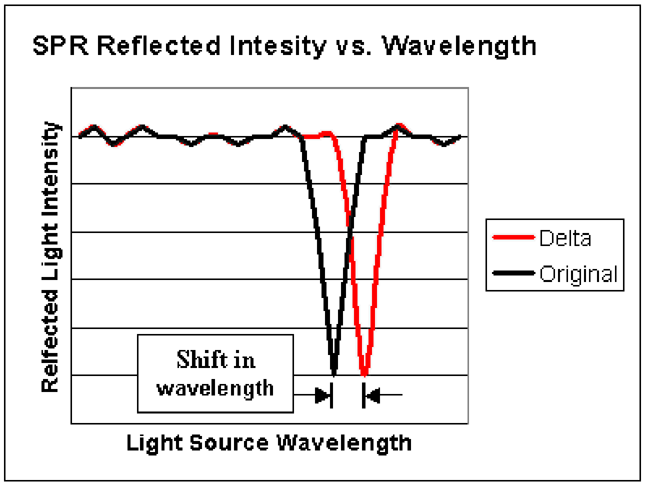


Figure 4. Depiction of measurable shift in SPR curve

3.2 THE SPREETA DESIGN

The biosensor component that meets the system requirements is the Texas Instruments Spreeta. It is an industry-standard component that is readily available for bulk purchase, thereby satisfying the low cost goal. The battery in a wireless device meets the Spreeta’s power demands thereby satisfying the portability criterion. The Spreeta is also satisfactorily rugged since it requires no delicate calibration, has no moving parts and imposes no constraint on orientation relative to gravity.

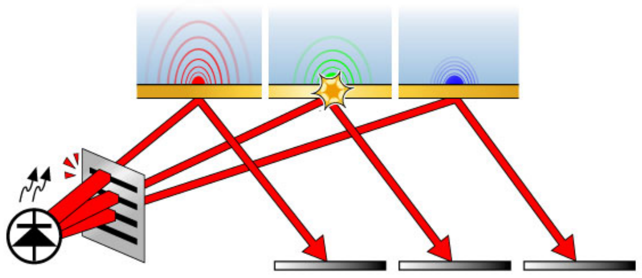


Figure 5. Spreeta's multiple SPR interrogation design

As shown in Figure 5, the Spreeta device has a rather ingenious design based on a single light source shining on the entire gold surface and reflecting onto an array of 128 light sensors. Thus a single Infrared (λ **= 840nm**) LED is the source for the Spreeta's entire range of SPR interrogation wavelengths. The gold film is tilted relative to the light source; this tilting results in a continuous change in the angle of incidence along the face of the film. When Transverse Magnetic (TM) polarized light strikes the film at an angle, the component of the wavelength that is parallel to the film is projected onto the film's surface.

An interrogation field induced by angled incidence is identical to that induced by perpendicular incidence at the projected (shorter) wavelength. Thus with a simple application of trigonometry it's easy to visualize that the steepest angle of incidence – generated at the point on the film closest to the light source – projects the longest interrogation wavelength (analogous to the "red-shift"), and the shallowest angle of incidence – generated at the point on the film farthest away from the source – projects the shortest interrogation wavelength (analogous to the "blue-shift").

CHAPTER 4: MONOLAYER (BIOLGICAL INTERFACE) DESIGN

The goal of the immunoassay design was to build a consistent biological layer of antibody attached to the gold surface of the Spreeta device. It is imperative to build a consistent layer across the entire gold surface because the analyte binding concentrations must be constant for every frequency. Given the portability of the sensor and supporting system, it was no problem at all to take the entire system into the biology lab to run the assays for the various experiments to be performed.

4.1 MONOLAYER COMPONENET SELECTION

We choose florescent analytes because they can be cross-correlated with other methods to better characterize the results. The antibodies, on the other hand, should be readily available and have a strong affinity to an analyte that's tagged in some way. The selection for the analyte was fluorescein isothiocyanate (FITC) because it fluoresces under ultraviolet light and therefore bound analyte to antibodies attached to a surface will remain after washing away un-bound analytes, then the monolayer can be examined for it's intensity of florescence. Further, using a Zeiss LSM510 confocal fluorescent microscope we can detect individual analyte attachment, which further clarifies the density of their attachment. With these cross-correlating methods, the immunoassay attachment results recorded by this biosensor design can be validated for consistency, repeatability and accuracy.

The antibody for FITC, anti-FITC is readily available and well characterized. Thus the combination of FITC and anti-FITC would have been good for sensor validation except that FITC is very hydrophobic and therefore does not stay in solution very well.

Another analyte with similar binding affinity to FITC is urinine (aka. Fluorescein sodium salt), which is also much more hydrophilic than FITC. Using FITC, which is not water-soluble would have forced us to use an organic solvent, which could result in denatured antibodies [3].

4.2 MONOLAYER IMMOBILIZATION

The procedure to prepare the surface, apply the antibody and add the antigen is outlined below. Basically, the surface is prepared by adding ions, which adhered to the gold, and then applying a mixture of antibodies (anti-fluorescein isothiocyanate; aka. anti-FITC) attached to the "Protein "A"" binding sub-assembly. Upon application of the mixture to the ion-prepared surface, the surface was a molecular stack-up of:

Gold Surface (Au) => Ions (H^+, OH^-) => Protein "A" => Antibody.

Once the analyte (Urinine) was added to the surface and a binding event occurred, the molecular stack-up of material on the sensor surface became:

Gold Surface (Au) => Ions (H+, OH-) => Protein "A" => Antibody => Antigen

Procedure to prepare the Spreeta surface:

1) Pipette 250 microliters (ul) of 1 Molar Hydrochloric Acid (HCl) onto the surface
2) Wash surface with water

   - Surface State: H^+ ions on the surface of the sensor

3) Pipette 250ul of 1M NaOH onto the surface
4) Wash surface with water

   - Surface State: OH^- ions added to the surface with H^+ ions

5) Pipette 250ul of TAE onto the surface

6) Wash surface with water

   - Surface State: OH^- & H^+ ions in TAE Buffer

7) Bake sensor at 70^oC for 30 minutes (until dry)

8) Weigh out 0.4mg of "Protein "A""

9) Dissolve 0.4mg of "Protein "A"" in 100ul of TAE

10) Agitate mixture for 20 seconds (using "Vortex Genie 2")

11) Pipette 15ul of fluorescein isothiocyanate (anti-FITC) into mixture

12) Agitate mixture for 20 seconds

   - Result: a 100ul vial of antibody attached to "Protein "A""

13) Pipette 250ul of antibody/Protein "A" solution onto sensor surface

   - Surface State: Antibody bound to "Protein "A"" bound to OH^- & H^+

14) Flick off all excess fluid on sensor surface

15) Pipette 250ul of $2.411x10^{-7}$ molecular weight fluorescein to the surface

   - Surface State: antigen now binding to antibody in real-time

4.3 MONOLAYER VERIFICATION PROCEDURE

The test procedure was to build the immunoassay on the Spreeta's gold surface and introduce the analyte while running the biosensor in real-time. With this procedure, it would be possible to:

a) Record the binding of Protein "A" to the sensor surface

b) Record the binding of the antibody to Protein "A"

c) Record the binding of the analyte to the antibody

Each of these biding event should cause changes in the permittivity of the material within the interrogation range. Binding events of any bio-molecule should cause a gradient change at gold surface and thereby shift the concentration of free ions in the interrogation range, which will register as a change in permittivity, measurable by the biosensor.

So the expectation is to run the biosensor during all phases of immunoassay, from “build to bind” and establish quantifiable metrics not only of how well the immunoassay is binding analyte, but also of the quality of the monolayer itself. For example, a poor binding density of Protein “A” would necessarily result in a poor antibody binding density. Thus this design allows qualitative analysis of the actual building of the monolayer in real-time. This design also provides closed-loop analysis for exploration of factors that might improve the various binding events throughout the build process (e.g. it would be possible to quantify the changes in binding density rates while adjusting temperature, thereby allowing the user to establish the optimum temperature for each step).

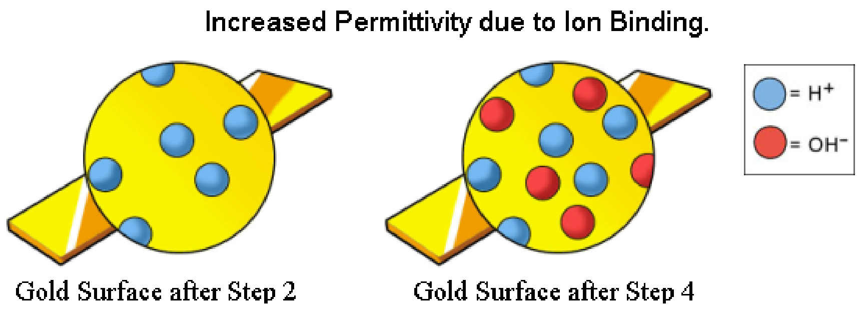

Increased Permittivity due to Ion Binding.
= H+
= OH−
Gold Surface after Step 2
Gold Surface after Step 4

Figure 6. Illustration of Permittivity Change Measurable During Monolayer Assembly

As illustrated in Figure 6, the ion density attached to the gold surface would be an excellent candidate to measure via a change in permittivity. Note that both H^+ and OH^- are free ions and thus both will cause increases in permittivity when increasing in the interrogation region near the gold surface. In a similar fashion, it would be possible to measure the density and even the rate of change of all materials attaching to the gold surface. As suggested above, the Protein “A” binding density on the gold surface could also be measured by skipping the pre-binding steps between the antibody and Protein “A” and simply introducing un-bound Protein “A” in solution. Once the binding density for Protein “A” in isolation has been determined, the pre-assembled Protein “A” and antibody complex can be introduced with a known value for the maxim binding rate of Protein “A” which can be used as an upper limit for the binding of the entire assembly.

The next two chapters: CHAPTER 5: HARDWARE DESIGN and CHAPTER 6: SOFTWARE DESIGNare computer engineering chapters describing the details of the sensor interface and algorithm design. CHAPTER 7: BASIC SYSTEM FUNCTIONAL VERIFICATION is a return to the general system. Readers interested in the biological and system aspects of this research may wish to skip over the computer centric details.

CHAPTER 5: HARDWARE DESIGN

The hardware requirements were further refined upon inspection of this data sheet for the Spreeta. It immediately revealed several important ramifications for any design that would incorporate it:

1) There are no moving parts on the Spreeta device
2) The Spreeta device utilizes a synchronous serial interface
3) The Spreeta device requires real-time performance to read the data
4) The output of the Spreeta device is analog
5) The data requires extensive smoothing and graphing manipulations

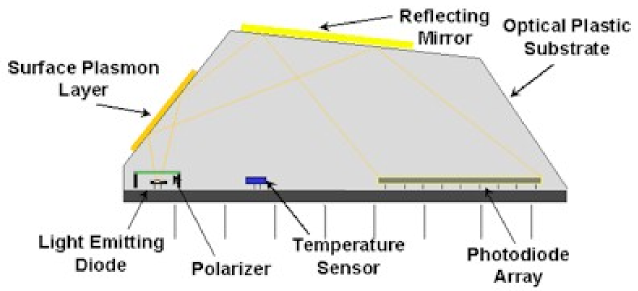


Figure 7. Internal Design of Spreeta Sensor

The lack of moving parts was more of a surprise than a design constraint, but it certainly makes the part easier to manufacture, more robust and lower cost (the Spreeta device is $30 in quantities of 100). However, this does lead to a system constraint that if there are no moving parts, each element in the photodiode array corresponds to a fixed angle. Thus the device becomes very easy to work with: simply read the intensity of a particular element and you immediately have the reflected light intensity at a given angle; there's:

a) No need of a control to advance or step through angle(s) of rotation,

b) No introduction of error due to a mechanical rotation of a mirror or prism,

c) No constraints for orientation relative to acceleration (e.g. acceleration would certainly affect moving parts)

As shown in Figure 8, the photodiode array consists of 128 elements, each of which is attached to the output in sequence. Every pulse on the clock pin connects the next pixel to the output. The light intensity is captured during the integration period and transferred to a Charge Coupled Device (e.g. a capacitor) for driving to the output stage at the next pulse on the start (SI) pin. This introduces two important timings:

d) the time between begin integration and end integration will determine how much light will be collected by the pixels in the array

e) the CCD is a leaky storage element thus the data must be collected from the device before non-negligible error is introduced into the results

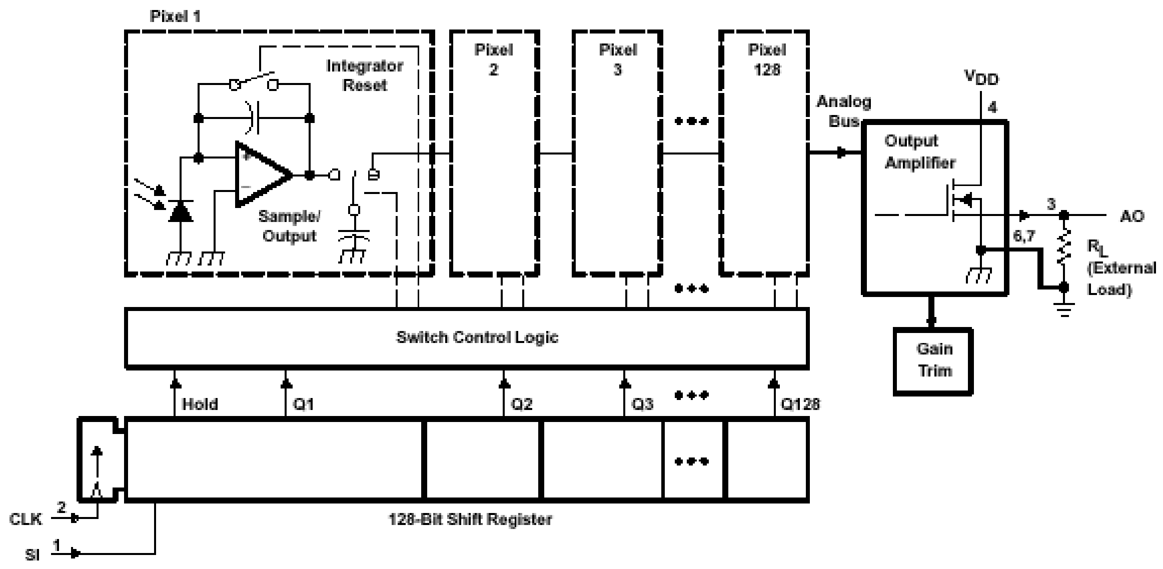


Figure 8. Pixel Array Design Internal to Spreeta Sensor

Thus one critical system constraint of the timing is to ensure the integration period never changes. Additionally, the LED must be tuned for the integration period: short integration period requires a more power to the light source (brighter); longer integration period requires less power to the light source (dimmer). In fact, if there’s too much power in the light source for the integration period, the outputs tend to saturate. Further, the specification for the array suggests the average value of all pixels should be around 2.5V.

5.1 SUPPORTING CIRCUIT COMPONENT SELECTION

Once the Motorola Accompli 009 was established as the host, the next design step was the interface hardware between the serial port on the Accompli 009 and the Spreeta. One immediately obvious issue was the miss-match in voltage levels: the Accompli 009 uses 3.0V CMOS rails and the Spreeta uses 5.0V TTL. One solution to this issue is a voltage shift register, but rather than burden the device with the cost, power, timing delay and real-estate of additional hardware the issue was resolved using an HCMOS device running at 4.0V to sit between the 3.0V CMOS and the 5.0V TTL, thereby saving the voltage shifters at the expense of an additional voltage rail.

In terms of the power sources, an advantage of two separate voltage rails (4.0V and 5.0V) was noise immunity. Given the analog nature of the design, it is critical to minimize the electrical noise to maximize the sensitivity of the Spreeta's output. Thus driving the Spreeta's power from a completely autonomous voltage source from the A/D converter was a significant noise immunity advantage in the design. Finally, to further insure low noise on the voltage rails the design incorporates linear (as opposed to switching) voltage converters. This constrained the voltage input to the interface board to be greater than 5V, and for simplicity the design incorporates a +12V DC supply (also readily available in the industry), which interfaces to the board through a diode bridge to yield the robustness feature of immunity to tip polarity.

The analog output of the Spreeta connects to an A/D converter. The design incorporates a converter with 12-bit resolution, again to increase sensitivity and a synchronous serial interface for compatibility with the SPI interface. The output voltage range of the Spreeta is specified as 100mV to 3V. The best configuration for an analog input to an A/D converter is to correlate the maximum value of the A/D output to the maximum voltage of its analog input. Thus the design incorporates an A/D converter that accepts an external reference. By using 3.0V as the reference the ranges are:

Spreeta output of 3.0V => 0xFFF on the converted digital reading

Spreeta output of 0.0V => 0x000 on the converted digital reading

Another challenge was to design an interface to the two serial components with only one chip select (the Accompli 009 provides only 1 chip-select) so the interface exploits a “synchronous bus expander” to interface between the two serial components (A/D and Spreeta) and the SPI port. Not only did this interface yield a critical voltage level step to help solve the power matching difficulties outlined above, it gave the design a lot of flexibility in controlling the A/D converter in conjunction with the Spreeta device. The design needs enough flexibility to run the basic algorithm:

1) Clock a “pixel” (e.g. a single CCD cell in the array) from Spreeta

   1a) for the first pixel, assert the Start pin to end the integration period

2) Read the digitally converted value of the pixel

3) Repeat for all 128 pixels in the array

The last component to select was the cable to interface between the Accompli 009 and the prototype board. This proved to be quite a challenge, as the interface had never been used before. Thus the cable was full custom and had to be built by hand. Knowing the propensity of a prototype cable to take a lot of strain (oops! forgot to unplug the darn thing) the cable’s design includes some strong, clear, non-conductive epoxy poured all over the junction between the wires and the connector headers.

5.2 IMPLEMENTATION

Once all the components were acquired (mostly as free samples), the hardware implementation was done on a breadboard using through-hole parts because they’re easy to work with (with the exception of the 3.0V reference which wasn’t available in a through-hole package so it was soldered as opposed to bread-boarded). To further ensure nose immunity the design incorporates several different values of power filters (usually a decoupling capacitors behind a series resistor) to the power supplies and to the power input pins on every component.

5.3 HARDWARE DEBUG

The design also includes a couple of LED's to facilitate binary board probing, but these proved to be very power-hungry so they were always disconnected during sensitive testing and definitely during the actual immunoassay. But the main debug tool was a logic analyzer, which is invaluable during real-time debug. It was used to verify the timing of the signals on the SPI interface as well as timing the software algorithms.

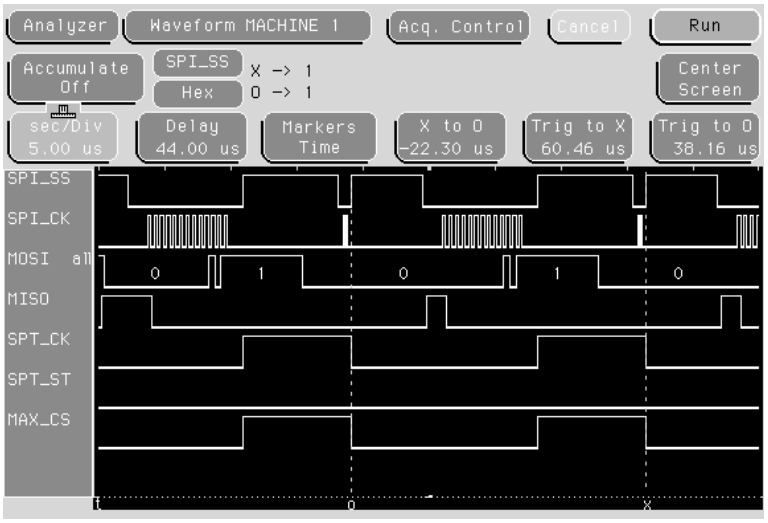

Accumulate Off
SPI_SS Hex X -> 1 O -> 1
sec/Div 5.00 us
Delay 44.00 us
Markers Time
X to O -22.30 us
Trig to X 60.46 us
Trig to O 38.16 us

Figure 9. Logic Analyzer Hardware Debug Trace

Some of the discoveries during the hardware design phase were a result of the Spreeta specification missing some critical information. The technical support folks at Texas Instruments provided the specification for the TSL1401 128x1 linear sensor array, which is the CCD component used internally by the Spreeta:

- The Analog output of the Spreeta device requires a pull down resistor

- The best values for the Potentiometer limiting power to the LED had to be experimentally sized as there was no data available on it's power
- The Spreeta is fast! Its analog output settles in 350ns

One of the decisions for this design was that the data stored in Spreeta's internal ROM not be accessed. This ROM contains data about the specific Spreeta device, which would be useful for guaranteeing that every device responded according to a standard (e.g. normalizing the output of all devices to eliminate manufacturing variances). Rather than normalizing between components by reading the data and adjust the Spreeta output, this design utilizes the innovative "differential mode" in which manufacturing variances are eliminated programmatically for a real-time analysis.

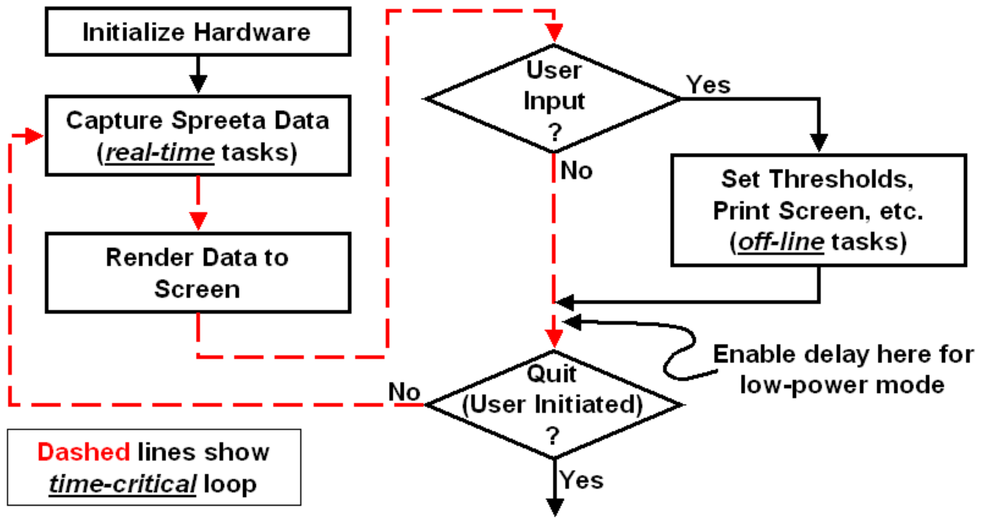
CHAPTER 6: SOFTWARE DESIGN

There were two main aspects to the software design: the algorithms for reading and manipulating the Spreeta data, and the User Interface. Thus the biosensor software had to keep the data moving from the Spreeta device to the screen while maintaining a screen update that showed the real-time aspects of the sensor's operation and responded quickly to user input. The basic challenges in the software design were:

- Maintaining real-time control of the Spreeta's light integration period in its photocell array
- Processing the data from the Spreeta using strictly integer calculations
- Defining a presentation mechanism with run-time analysis that yields instantaneous operational results and simple limit detection
- Maintaining a continuous update on the screen

- Receiving and processing user inputs for control with minimal delay

The data manipulation software was designed with a hierarchy of tasks broken up into three categories: real-time, time-critical, and off-line (see Figure 10). The real-time tasks are defined by the sensor hardware: the integration periods of the A/D converter and the Spreeta photocells. The time-critical tasks are defined by the “continuous operation” requirement and include screen updates and loop checks (e.g. check for user input). The off-line tasks are all tasks that suspend the continuous operation of the sensor. These are typically user initiated (print screen, set threshold, etc.) with “alert user of threshold violation” being an example of non-user initiated task.


Initialize Hardware
Capture Spreeta Data (real-time tasks)
Render Data to Screen
User Input ?
Yes
No
Set Thresholds, Print Screen, etc. (off-line tasks)
Enable delay here for low-power mode
Quit (User Initiated) ?
No
Yes
Dashed lines show time-critical loop
Figure 10. Main software loop

6.1 REAL-TIME TASKS

The Motorola Accompli 009 has a real-time operating system that supports precise timing for the light integration period in Spreeta’s photocell array and supports dedicating a hardware timer to the task. The timer period can be adjusted via user input during normal operation, which sets the average value of all pixels. Adjusting the integration period also has the effect of moving the entire waveform horizontally within the graph:

A shorter integration period => shift down in the pixel's absolute values

A longer integration period => shift up in the pixel's absolute values

To maximize the precision of the integration period, the algorithm shuts off interrupts, waits for the user-programmed delay, then enters a loop of 128 cycles of clock a photocell value to the A/D converter and read the A/D value (see Figure 11). The delay clock resolution is 1µs, which is sufficient to minimize the waveform jitter between cycles. The delay is initialized to 5ms (5000 ticks) and can be incremented and decremented in 50µs increments during normal operation. Note that the timer's minimum resolution is 30ns but clocking faster increases power consumption while only producing negligible improvements in jitter.

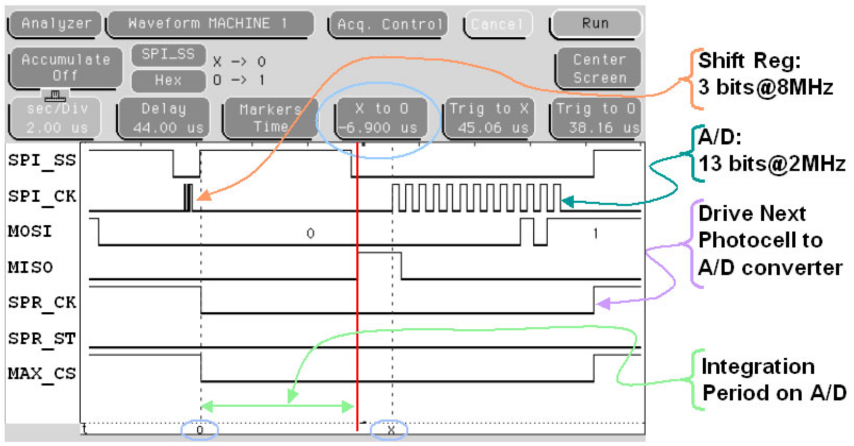


Figure 11. Logic Analyzer Capture: real-time control of Spreeta, A/D converter, and Shift Register on the SPI bus.

One of the foremost optimizations for the software was the minimization of the time required to cycle through the real-time loop. Unloading the 128 photocells from the Spreeta as quickly as possible after integration minimizes any effects of voltage leakage in the Charge Coupled Devices (CCDs) used to store the collected light. The result of these optimizations resulted in a Spreeta clock rate of 45KHz, which is well below he minimum value specified for the array of 5KHz.

6.2 TIME-CRITICAL TASKS

The Motorola Accompli 009 provides programmable access only to the display processor; the DSP is dedicated to wireless communications and is therefore not accessible by user-loaded programs. Additionally, the display processor is sized for low cost and minimal power consumption, not intensive floating-point calculations. This architecture imposes the constraint of "integer only" math because even a simple floating point add takes multiple hundreds of machine cycles at 33MHz and operations like waveform smoothing and screen mapping requires multiple floating point operations per pixel.

Since one of the design goals is to maintain a fast screen draw to track the Spreeta's activity, the algorithms were designed to eliminate as much floating point processing as possible. One example is the waveform-smoothing algorithm, which requires only shifts and adds: the array of 128 light intensity values (I[128]) that correspond to 128 different wavelengths projected onto the gold film, were smoothed by:

I[N] = (((I[N-1] + I[N-2] + I[N+1] + I[N+2]) >> 2) + I[N]) >> 1;   (**Equation 1)**

After smoothing, the intensities are mapped directly onto the absolute graph which is 128 pixels wide yielding a 1-to-1 correspondence between intensities and the screen pixels along the X-axis. The Y-axis is a voltage scale and thus requires a conversion to the screen height (see Figure 12a). Floating point calculation was unavoidable for this case but divides were converted to reciprocal multiples, which saved thousands of processor clocks per calculation and yielded a rate of approximately 8 full cycles/second.

The final challenge was detecting shifts in the absolute minimum of the graphed waveform; shifts in the minimum represent changes in permittivity at the sensor's surface; a change in permittivity at the sensor's surface is the physical phenomenon intended to be caused by molecular events occurring in real-time. Thus the system must have an optimization point to both detect and display in obvious graphical methods the shift in the absolute minimum of the array of pixels generated by the Spreeta device.

The first solution was to allow the capture of an “initial” waveform and superimpose subsequent waveform cycles in red. This had the desired effect of showing shift continuously over time, but with a minimum resolution of 1 graphical pixel on the screen. This led to a novel solution for displaying and amplifying SPR shifts without requiring floating point calculation: the definition a new waveform format called “differential mode” (see Figure 12b).

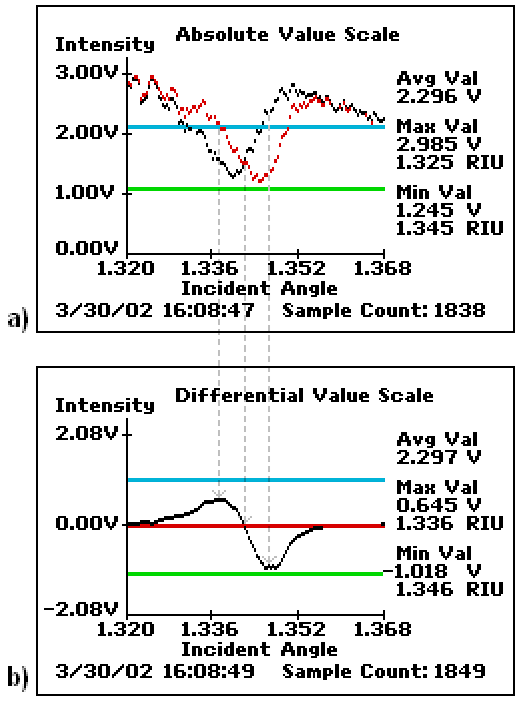


Figure 12. Biosensor screens illustrating the relationship between the "Differential" and "Absolute" modes

The differential waveform is generated by a simple 128-cycle loop, which subtracts the "New" values from the "Original" values. After calculating this difference the algorithm simply plots the difference values to the screen. This simple graphical presentation method can be exploited as a powerful mechanism for continuously monitoring an SPR shift occurring in real-time. It also yields:

a) A simple check for changes over time by marking the height of the differential curve over the "zero level"

b) Enforcement of a threshold with a simple check against the maximum and minimum threshold values

c) Auto-correlation of Spreeta output that normalizes waveform differences due to device variation

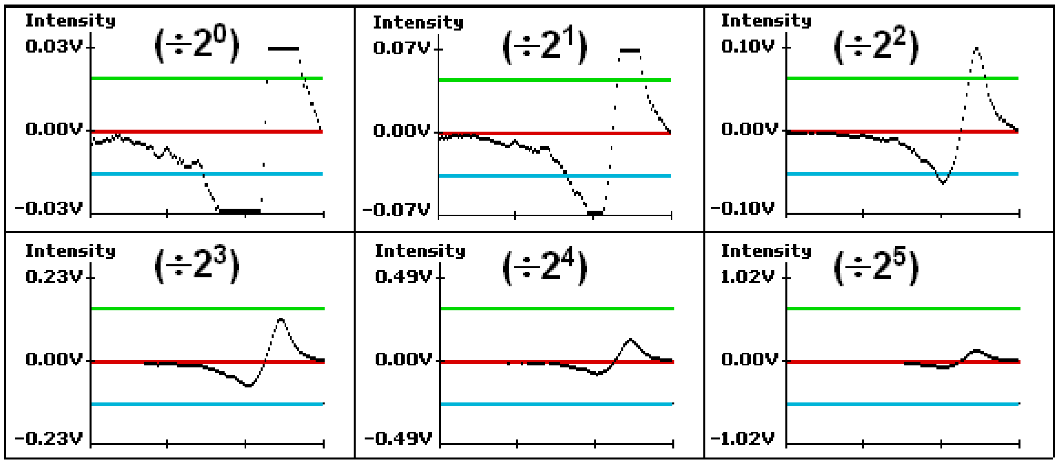


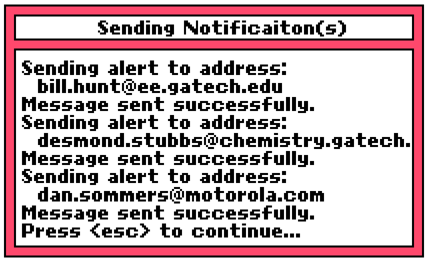
Figure 13. Differential Screen Progression showing the same waveform magnified (divided by) increasing powers of 2

A final advantage of the differential format is that we can control the values on the Y-axis. By selecting values that were an even increment of a power of 2, the algorithm is capable of mapping the voltages to screen values by a simple left-shift (e.g. voltages/2^1 = voltages>>1; voltages/2^2 = voltages>>2 … voltages/2^9 = voltages>>9; 2^0 = no shift at all, see Figure 13). Note that the voltages (intensities) may well be greater than the values at the Y-axis in which case the algorithm simply "clips" the data and plots the maximum height for the scale. This simply represents saturation for the chosen display limits but is no loss of data: simply dividing by a greater power of 2 will allow the data to be represented meaningfully on a graph with a broader range.

By completely eliminating floating-point calculations in differential mode, the main loop runs at approximately 23 full cycles/second (a 3x reduction in cycle time), which results in a faster screen update. Thus the fastest way to realize the real-time results of the Spreeta's operation is to choose the differential view.

6.3 OFF-LINE TASKS

The off-line tasks suspend the normal operation of the sensor, perform the required task, and then resume normal operation. One of the longest tasks is sending the e-mail notifications upon detection of a threshold violation (see Figure 14). This routine will encrypt the user-entered message content and send it to the e-mail destination address list wirelessly over the public networks; secure [14] data with guaranteed delivery.



```
Sending Notificaiton(s)

Sending alert to address:
  bill.hunt@ee.gatech.edu
Message sent successfully.
Sending alert to address:
  desmond.stubbs@chemistry.gatech.
Message sent successfully.
Sending alert to address:
  dan.sommers@motorola.com
Message sent successfully.
Press <esc> to continue...
```

Figure 14. Sending Wireless e-mail Notification

6.4 SOFTWARE IMPLEMENTATION

There are two main areas for the design of the software: the Spreeta driver with many real-time aspects and the display driver which is compute intensive and must accept user input. The challenge was to get these two environments to co-exist while meeting all of their diverse (often conflicting) requirements. Thus the software system must solve the following list of requirements:

1) the Spreeta device must be sampled at a fixed period
2) user input must be event driven for response time but serviced only outside the fixed period
3) screen updates must run either outside the fixed sample period or must be interruptible by the fixed period

Given this set of requirements, it was possible to design the flow of the main execution loop:

```
while (quit != TRUE)
{
    SampleSpreeta(BeginIntegrationPeriod);
    StartCounter();
    RepeatLoop(TotalPixelCount)
    {
        if (TimerExpired)
            SampleSpreeta(CaptureData);
        ProcessPixelValue();
        if (TimerExpired)
            SampleSpreeta(CaptureData);
        PixelToScreen();
    }
    Wait(TimerExpiration)
        SampleSpreeta(CaptureData);
    UpdateStatistics();
    GetUserInput();
}
```

Thus the main flow of execution starts with sampling the Spreeta data, which has the effect of beginning the integration period. The integration period must end at precisely the same time for every sample, or the light intensity readings delivered by the Spreeta will vary their intensity readings based on how long the light sensors have been exposed to the light. To control the timing, the system sets a timer running and ends the integration period by executing the sampling of the Spreeta data at the timer's expiration.

Note that the first sampling is taken during system initialization. By this method, the algorithm is always processing the previously captured sample while the Spreeta is integrating the next sample. Note also that while the Spreeta integration period is fixed, the time between samples can vary depending on user input.

6.5 USER INTERFACE

Since this is a fully functional Accompli 009 system, and the Spreeta has real-time requirements, it is necessary to halt normal operations of the system and dedicate the processor to running the Spreeta sampling algorithm. Navigating into the Spreeta application on the desktop and launching the biosensor driver accomplishes this halting normal operations. Upon launching the biosensor, the Accompli 009 will reset, and upon subsequent boot, will launch only the Spreeta algorithm.

Desktop Biosensor Launch Screen

Algorithm Initialization Screen

Future implementations could certainly have the user interface running in the native Accompli 009 desktop (written in a programming language called "FlexScript") and have the Spreeta driver running in real-time underneath the desktop. In this configuration, however, the screen updates wouldn't be as fast, but the rest of the system (e.g. GSM connectivity) would be available. This would be advantageous for communicating results and/or real-time detections of perturbations either by voice or automatically sending data to servers and/or system controllers.

The inputs from the user are the keypad, and the outputs are the screen for real-time observation of the system and the Infrared port for reading data off of the Accompli

009. For simplicity, the user interface is a simple single-keystroke menu with the command show in Table 1.

Table 1. List of Commands for Biosensor Control

• **S – Start running (only used at power-up)**
• **A – Render data in Absolute Draw Mode**
• **Z – Render data in Zoom Draw Mode**
• **D – Render data in Differential Draw Mode**
• **0-9 – Set vertical scale for Differential Draw Mode**
• **C – Capture baseline waveform to measure Differential**
• **J – move waveform down (decrease integration period)**
• **K - move waveform up (increase integration period)**
• **L – Low power mode (insert delay in main loop)**
• **H – High power mode (remove delay from main loop)**
• **P – Print screen to memory (for retrieval out IR port)**
• **Nav-Down – Move active cursor down one pixel**
• **Nav-Up – Move active cursor up one pixel**
• **Tab – make alternate cursor active**
• **N – enable wireless Notification (prompt for e-mail addr.)**
• **T – invoke Threshold (used mainly for test)**
• **Q – Quit running Biosensor program (return to phone…)**

Note that the user can type either upper case or lower case.

The graph on the screen is 128 pixels wide by 90 pixels high. The width was chosen according to the number of pixels across the angle sweep in the Spreeta device. The height was chosen as a matter of convenience: giving as much height as possible to the active area of the graph while leaving adequate room for the statistics. The screen layout is driven by constants so moving the location of screen objects is straightforward; by changing the definitions, the screen-draw algorithms update the data in the new locations.

The choices of screen draw mode (Absolute, Zoom and Differential) change how the sample data gets processed in the "**ProcessPixelValue**" call outlined above. For example, in the Absolute mode, the intensity reading read for each location in the angle sweep is mapped directly onto the screen with a 3.0V reading mapped to the top and a 0.0V reading mapped to the bottom. Unfortunately, this mapping requires floating-point divides and since there is no floating-point hardware support on the Accompli 009 user-interface processor, this calculation is actually the limiting factor in how fast we can run the sampling algorithm.

```
Draw_Location = (int)((Y_BOT_VERT_LABEL+TEXT_HEIGHT)
              - (int)((int)(gSpreetaPixelValue[pixel_count]
               -  gScreenMinValue) * gConvertADCToScreen));
```

The Zoom mode is basically the same as Absolute mode except that the vertical axes are adjusted so the minimum and maximum data values shown on the screen just fit within the definition of the axes extremes. For both modes, a data value is drawn as 2 pixels high in order to enhance the visual effect and to smooth out the graph just a little.

6.6 DIFFERENTIAL DISPLAY MODE

Perhaps core to the design of this sensor is the mode definition for differential. The Spreeta device measures changes in the permittivity of the biolayer by detecting the change in light intensity reflected at a given angle. Thus the amount of light energy absorbed by the gold layer changes for different wavelengths of light, and the wavelengths of light correspond to different angles of reflection against the gold surface. Unfortunately, the changes to the light intensity are quite small (less than a pixel at the resolutions shown on the screen) and the waveform on the screen is far from ideal.

To solve these difficulties, it is quite common to run through compute-intensive smoothing algorithms and detect local minimums and maximums, thereby detecting the absolute change in the waveform. However, given the lack of floating-point hardware on the Accompli 009 processor and the hardware limitation of not supporting reading the pixel variation data, running through smoothing algorithms isn't a viable option. Instead, detection is done simply by subtracting the original (captured) data from the most recent sample and graphing the difference between the two (See Figure 15).

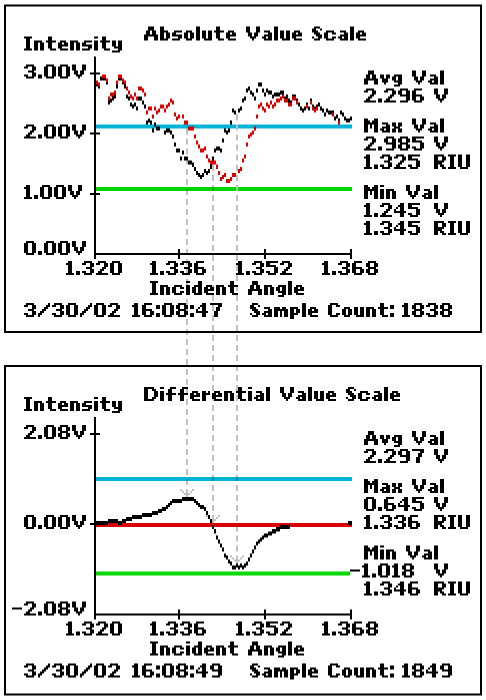


Figure 15. SPR Shift in Absolute and Differential Modes

As illustrated above, if the analyte attachment causes a shift in the light intensity read over a range of angles, there will be a shift to the right in the absolute waveform at the local minimum for plasmon resonance. However, the shift will not be as pronounced as shown. As stated above, the shift will be less than one pixel. Thus the differential screen has a vertical axis that is very narrow, reacting to slight changes in the intensity reading from the Spreeta sensor. To avoid floating-point calculations, the intensity on the vertical axis is 8 A/D ticks per pixel: thus a change in light intensity reading of 5.86mV will cause a 1-pixel offset in differential mode.

Note that in differential mode, the red line across the middle of the graph represents that for a given pixel the captured intensity is the same as the newly sampled intensity. This has the effect of self-adjusting for the pixel differentials across the pixel array by displaying only the change of a given pixel. Thus if a pixel is consistently reading high, then:

HighValue - HighValue = 0 (Equation 2)

Alternately, if a pixel is consistently reading low, then:

LowValue – LowValue = 0 (Equation 3)

So only a change is recorded on the differential screen:

pixel offset = (captured pixel value – sampled pixel value) / 5.86×10^{-3} (Equation 4)

As long a each pixel is linear with respect to itself, this algorithm will show the changes to a given pixel without requiring any interaction between the pixels. As the plasmon resonance causes the angle of maximum absorption to shift to the right, the differential screen will move farther off of the 0-differential axis. The height above the axis is an indication of how much the angle has shifted (as shown in Figure 16).

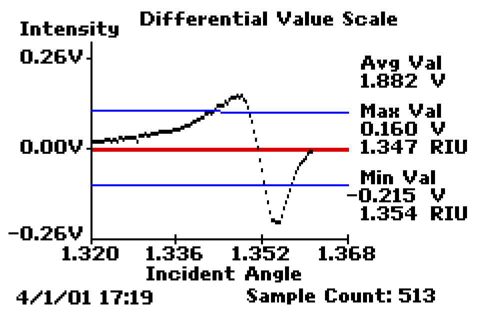


Figure 16. Height Of Differential Curve Above & Below Thresholds

6.7 SPREETA DRIVER

The integration period of the Spreeta samples had to be carefully controlled as outlined above, but another constraint of the Spreeta driver is how fast it gets read. Since the Spreeta is based on Charge Coupling (and therefore the charge can leak away) the data captured in the Spreeta is only guaranteed to be valid for a short amount of time. Thus the read algorithm had to be run as fast as possible. To accomplish this task, the device is read in a tight loop:

```
Repeat (for all pixels)
{
   SetSPIHardwareForMaxim();  /* 12 clocks at 4 MHz */
   ReadA/Dvalue();           /* read Spreeta Pixel */
   SetSPIHardwareForSpreeta() /* 1 clock */
   ClockSpreetaDevice();      /* advance Spreeta Output */

}
```

This algorithm also relies on a 7.1uSec settling time for the A/D converter to sample its input. The sampling time is controlled by the extra clock cycles in the processor to repeat the loop. Also, the pseudo code for "**SetSPIHardwareForMaxim**" and

"`SetSPIHardwareForSpreeta`" includes the control to the 74HC594 shift register to generate the appropriate chip selects to the A/D converter or the Spreeta device. I spent quite a bit of time tweaking and minimizing the sampling loop to achieve optimum performance. The minimum specification for reading the Spreeta pixel array is 5KHz.

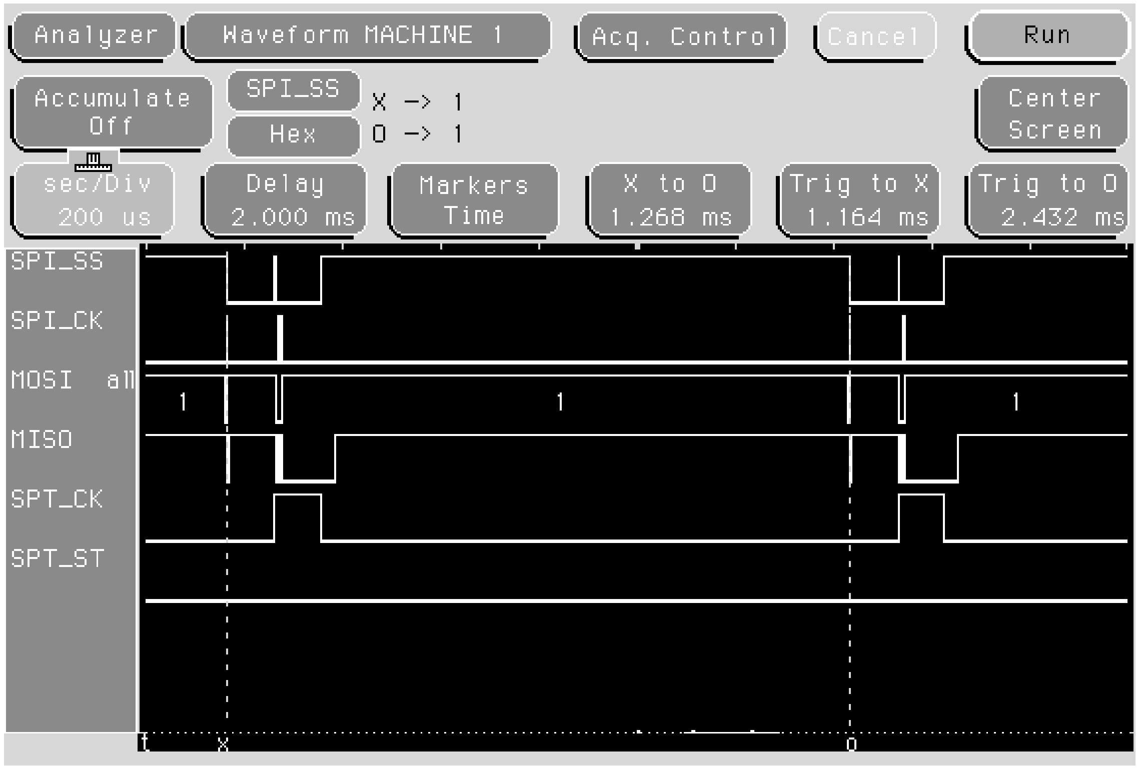


Figure 17. Sample Rate: 790Hz

As shown in Figure 17, the initial rate for sampling the Spreeta device was reading each pixel in the array at 790Hz. This rate is below the minimum rate for the Spreeta specification, so optimization of this loop was mandatory. Applying real-time analysis with the logic analyzer revealed that there was a significant amount of time being spent on the floating-point conversions identified the first optimization. The processor in the Accompli 009 does not include a floating point processor so even simple conversions of a voltage to a screen position is a significant amount of real time. Thus floating point calculations were either eliminated with manifest constants or converted from divides (thousands of cycles) to multiples (hundreds of cycles) by multiplying by the reciprocal instead of straight division.

```
screen_pixel_value = (int)((BOTTOM_GRAPH_BOUNDARY)
                     - (Int)((int)(gCurrentSpreetaValue[pixel_count]
                     - gScreenMinValue) * gConvertADCToScreen));
```

These conversions resulted in a reduced pixel read timing of 2.9kHz as shown in Figure 18.

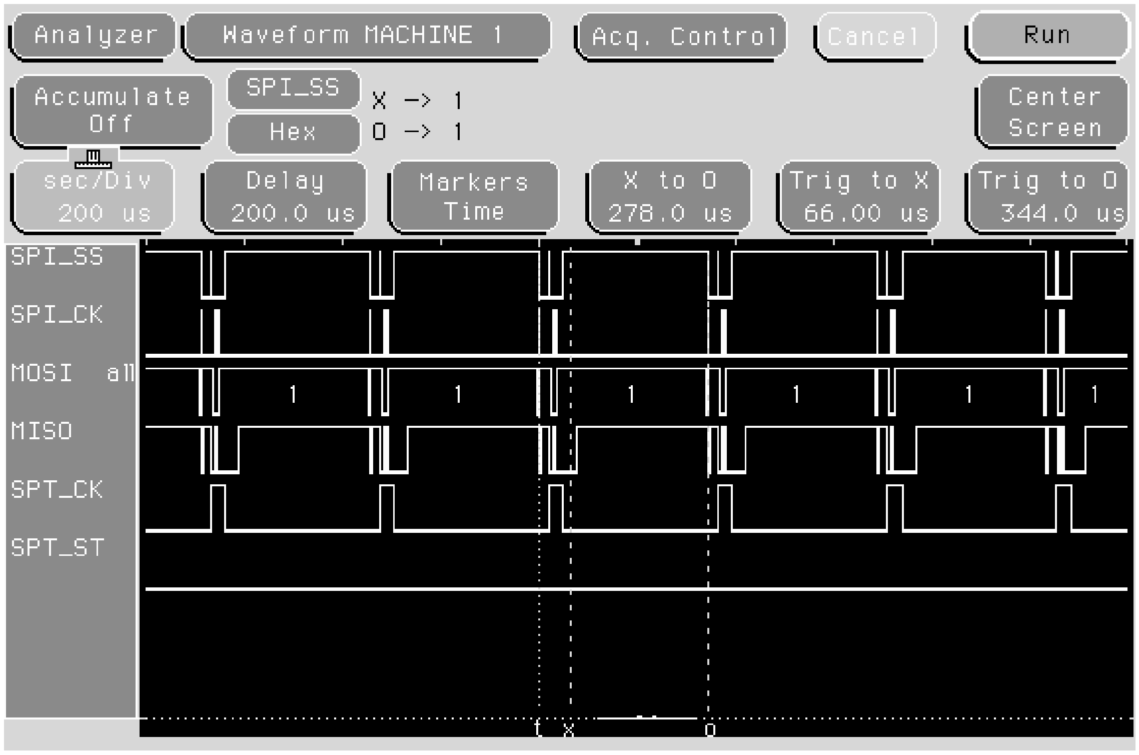


Figure 18. Sample Rate: 2.9KHz

A sample rate of 2.9kHz is certainly an improvement but still not quick enough to meet the timing requirements of the Spreeta. The final optimization was to change the basic algorithm so that instead of reading a pixel from the Spreeta and then writing it to screen, all the pixels were captured in a tight loop, then written to the screen when the timing wasn't nearly so critical.

```
MASK_INTERRUPTS;
START_COUNTER;
SampleSpreeta(BEGIN_INTEGRATION);
wait(gIntegrationPeriod)
```

```
    SampleSpreeta(CAPTURE_PIXELS);
RESTORE_INTERRUPTS; /* re-enable interrupts */
RESET_COUNTER;      /* turn off the counter to save */

/* draw the samples to the screen */
for (pixel_count = 0; pixel_count<TOTAL_SPREETA_PIXELS;
pixel_count++)
{
PixelValue = ProcessPixelValue(pixel_count,algorithm_mode);
PixelToScreen(PixelValue,pixel_count,algorithm_mode);
}
```

With this change, the timing sped up significantly to 45kHz as shown in Figure 19.

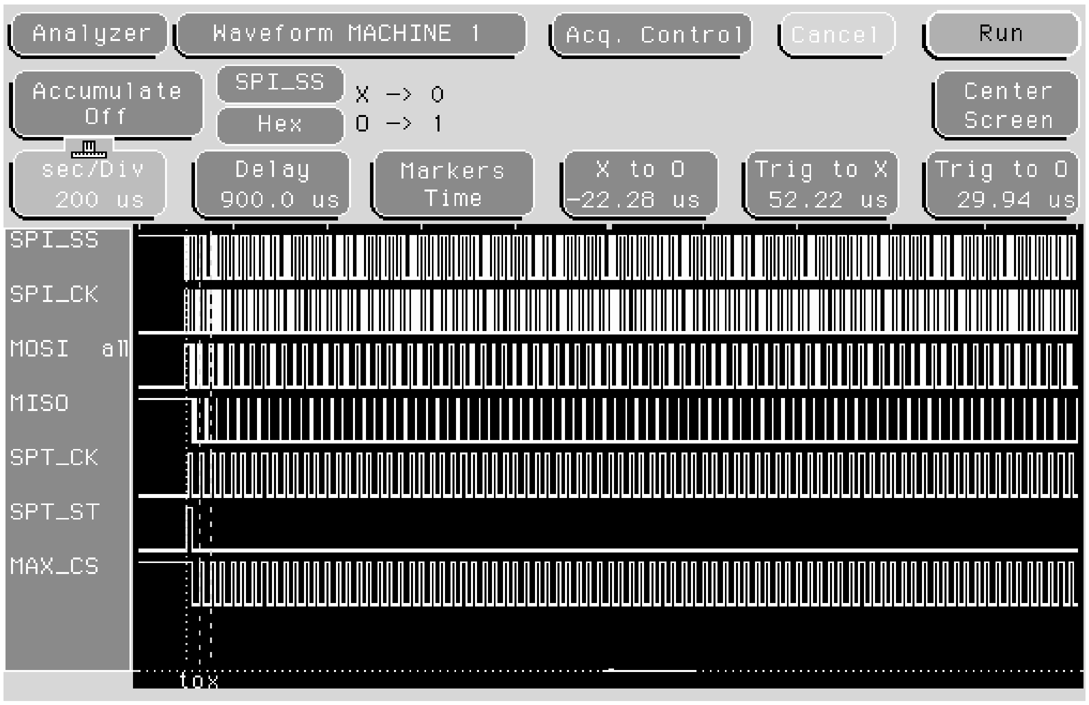


Figure 19. Sample Rate: 45KHz

Thus Discoveries made during the software development phase include:

A) Integration time for the Spreeta must be a fixed, non-varying value; even a single floating-point multiply is thousands of CPU cycles and thus floating-point must be avoided where performance is an issue

B) Reading the pixel array must be protected from all variation in timing (e.g. turn off interrupts while reading the array)

The total time to read the Spreeta array is only 128 Samples/45KHz = 2.8mSec, which is short enough in the Accompli 009 to be comfortable with disabling interrupts. Thus even in a fully operational environment, this driver can be event driven on a timer, and when the timer fires the event handler could certainly turn off interrupts, run the sample routine, then return control back to the higher-layer processes for running the full Accompli 009 system.

6.8 WIRELESS NOTIFICATION

Because one of the system requirements was ***Autonomous Action:*** (see requirement number “5)” on page 5) the sensor is designed with the capability to generate notifications upon detection of threshold violations. The use case is to allow an operator to set a limit threshold of how much the SPR curve is allowed to shift, and enable notifications.

6.8.1 NOTIFICATION ALGORITHM DESIGN

Wireless notification is one of the intrinsic capabilities of the Accompli 009 but the processor is consumed while that notification is being sent over the air. To ensure smooth operation, a constant check for the crossing of the threshold has been enabled in the time-critical loop first discussed in section 6.2, “TIME-CRITICAL TASKS“ and in Figure 10. Main software loop”. As shown in Figure 20, enabling notification inserts an additional check in the time-critical loop called “Crossed Threshold”. Note that this does not affect the Real-Time Tasks within the “Capture Spreeta Data” block.

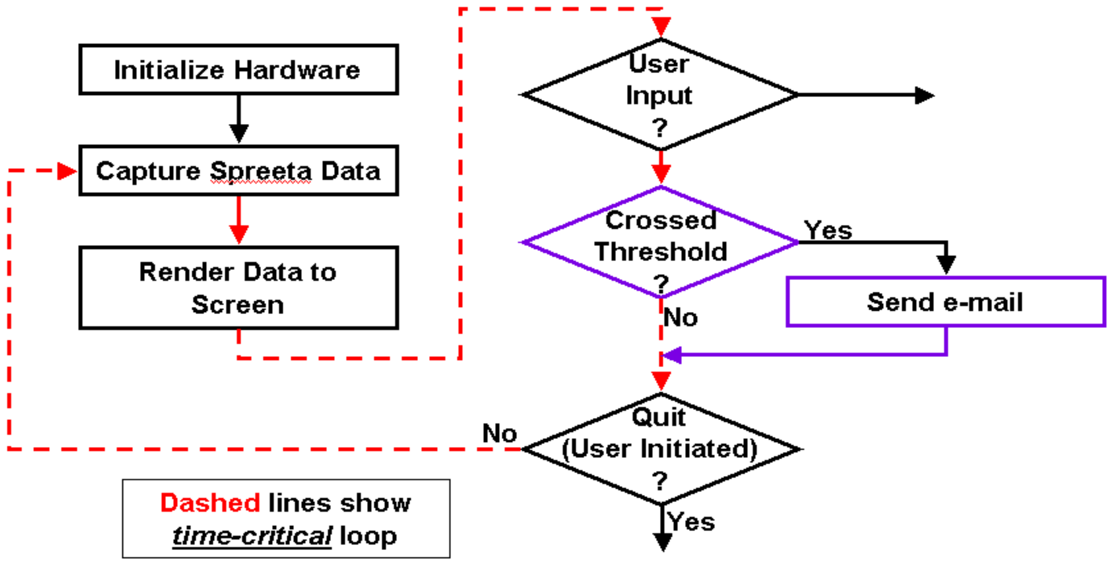
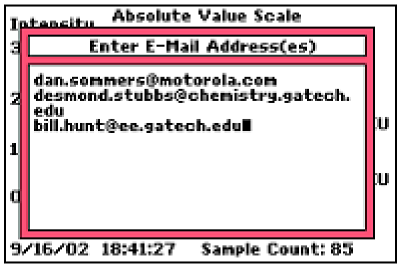


Figure 20. Extra test for threshold crossing when Notifications is enabled

Upon detection of the crossing of a threshold, the algorithm detours away from the main loop, and drops into a "Send e-mail" task. This task is actually one of the more challenging in that it is well designed for a multi-tasking operating system and thus event driven. However, since the main algorithm was implemented as a relatively simple state machine for performance reasons, the only state that normally handles events is in the "Check User Input" block. Thus this new e-mail task had to have it's own event handler and had to be interoperable with the user input event handler to continue to process key events while sending messages (e.g. "Cancel").

6.8.2 NOTIFICATION ENTRY DESIGN

The notification is intended to be prepared during system set-up and therefore has the luxury of sitting outside the "Time-Critical Tasks" since the device isn't expected to be sensing during set-up. During this set-up time, the operator will set the thresholds (both low and high) and enter the notification address and text. The user interface to enter notification addresses is shown in Figure 21. The design of the text box allows for as many recipients as the user desires to enter, separated by a carriage return (“enter” key on the Accompli 009). The e-mail address can be edited until they are all correct and then the user presses the “tab” key to advance to the next screen.



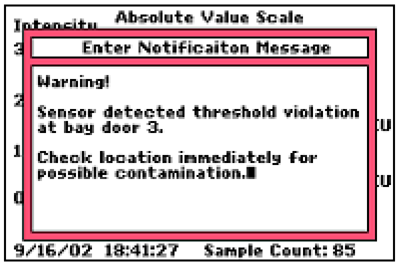
```
dan.sommers@motorola.com
desmond.stubbs@chemistry.gatech.
edu
bill.hunt@ee.gatech.edu
```

9/16/02 18:41:27 Sample Count: 85

Figure 21. User Interface to enter Notification E-Mail Addresses

The notification address follows the Internet e-mail address format and as such there’s no limit on where the notification can be sent. The recipient repository of many e-mail addresses commonly in use is machine-based and therefore capable of taking action as directed by the e-mail content. An example of this might be an automated help-desk or order entry system.

After entering the notification e-mail addresses, the entry design code presents the user with another dialog box in which the operator types in the actual notification text (see Figure 22). The notification text is free form and can be any set of characters the operator desires to enter. This allows the freedom to have human-readable text intermixed with machine-readable text. The machine-readable text must necessarily be keyboard entry and delimited (such as is typical in XML formats) but this constraint is not severely limiting to the capabilities of most automated e-mail services.



Enter Notificaiton Message

Warning!

Sensor detected threshold violation at bay door 3.

Check location immediately for possible contamination.

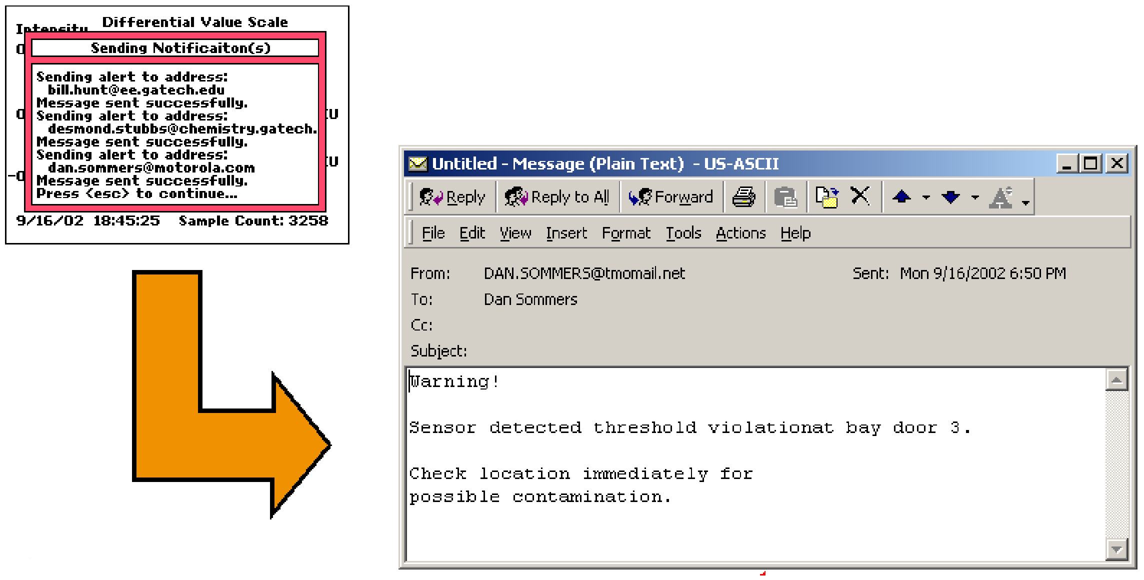
9/16/02 18:41:27 Sample Count: 85

Figure 22. User Interface to enter Notification Message

Note that the Accompli 009 is capable of sending binary data (as opposed to ASCII data) but the design doesn't support the entry of binary data from the keyboard. This capability could be added in any of several ways including download of messages via the IR port or defining some escape keys in the keyboard input to allow binary typing (rather tedious to type it, however).

6.8.3 SENDING NOTIFICATION

As illustrated in Figure 23, when the sensor detects a crossing of the threshold, it will automatically generate the wireless messages sent to each recipient in the notification list. There are error checks in place to verify that the e-mail successfully traversed the wireless network and it is possible to implement "read receipts" for further checks to ensure the recipients consumed the notifications (although this feature has not been implemented at the time of this writing).



```
Sending Notificaiton(s)

Sending alert to address:
  bill.hunt@ee.gatech.edu
Message sent successfully.
Sending alert to address:
  desmond.stubbs@chemistry.gatech.
Message sent successfully.
Sending alert to address:
  dan.sommers@motorola.com
Message sent successfully.
Press <esc> to continue...
9/16/02 18:45:25    Sample Count: 3258
```

Untitled - Message (Plain Text) - US-ASCII

From: DAN.SOMMERS@tmomail.net   Sent: Mon 9/16/2002 6:50 PM
To: Dan Sommers
Cc:
Subject:

```
Warning!

Sensor detected threshold violationat bay door 3.

Check location immediately for
possible contamination.
```

Figure 23. Notification Sent from Sensor to E-Mail

Note that the example shown has "**Press <esc> to continue…**" which would be unacceptable in an actual autonomous monitoring situation. The user interface has the capability to disable the requirement of a key input to turn return to monitoring. It is simply much harder to capture a screen shot without the pause in place.

6.8.4 NOTIFICATION SECURITY

One serious aspect of this type of wireless transaction is "spoofing". Since these addresses are wide open on the Internet, any e-mail system is capable of sending notifications to them. It is obviously unacceptable to be wide open to "spoofing" where any hacker with an e-mail account sends out warnings that the system under test has been compromised. The Accompli 009 has destination specific encryption capability using triple-DES algorithms and FIPAS 140-1certified key management strategies [18] that will ensure that the recipient can verify any messages have come directly from this sensor without any chance of being modified along the way. The security implemented by the Accompli A009 is part of the Motorola Messaging Platform (MMP) [19] and the basic architecture is as shown in Figure 24.

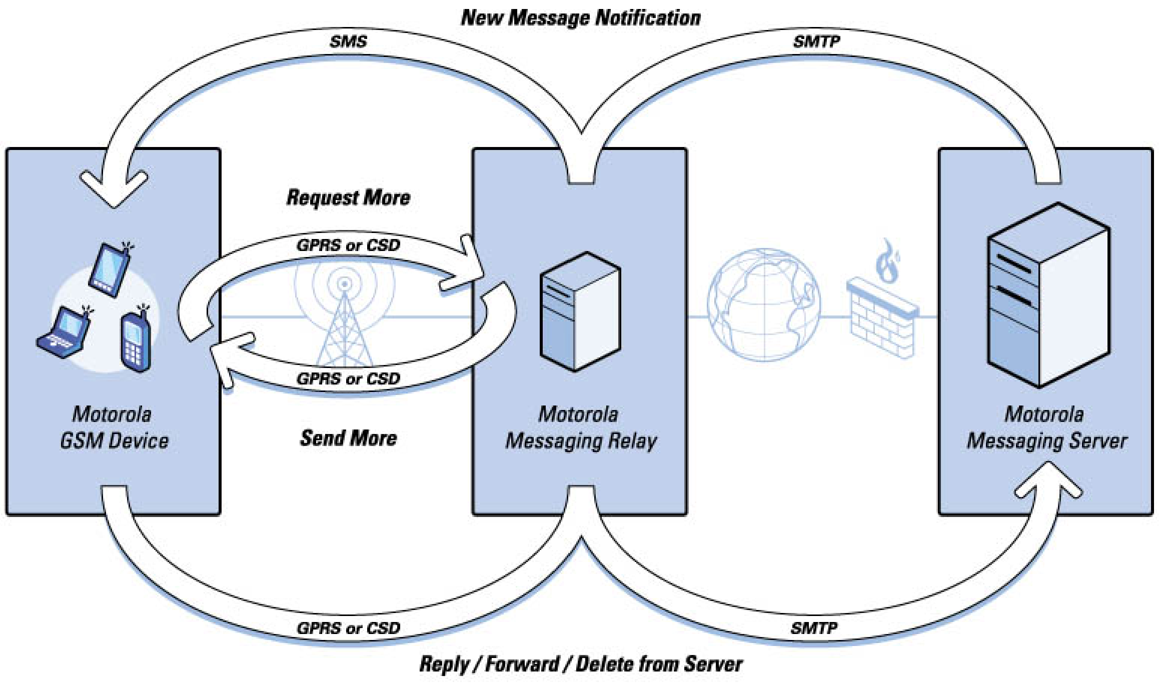


Figure 24. End-To-End Encryption in Accompli 009

CHAPTER 7: BASIC SYSTEM FUNCTIONAL VERIFICATION

Once the system was up and running with the hardware and the software, the next step was to connect up a fully functional Spreeta device to see what types of reading could be extracted from it. Note that for the initial hardware & software development phases, the only Spreeta component we had to work with was a mechanical sample that was functional but not expected to yield as strong results as a device that was considered fully functional.

7.1 LED BRIGHNESS AND LIGHT INTEGRATION PERIOD

The first step was to connect up the device and adjust the LED brightness against the integration period. To simplify initial calibration, I decided to keep a fixed integration period and adjust the LED brightness with a Potentiometer. By using an actual working sample, I found that when the integration period was too short, the LED needed to be adjusted to a very high intensity by adjusting the resistance in the LED's path to a very small value: on the order of 25 Ohms. I also learned that there is a definite upper-limit to the brightness of the LED, and adjustments beyond that limit cause the device to stop functioning.

At this point we requested additional samples from Texas Instruments (they graciously sent me 5, for free) and I resumed calibrating the system. The final choice for the two parameters has settled out to be:

1) Integration period default = 20mSec
2) LED resistance = 10Kohm Potentiometer + 67ohm fixed

Given this configuration, it is possible to adjust the potentiometer by hand and easily move the output waveform up to saturated (all outputs > 3.0V) or down to almost 0 with no shape to the waveform at all. According to the Spreeta spec. The best average value for all readings is 2.5V.

7.2 AMBIENT LIGHT

Another discovery during the system verification phase is that the sensor is very sensitive to ambient light. It was a bit of a surprise to see the entire waveform move up and down on the absolute scale depending on whether the sensor is in a dark environment or a lighted environment. Since I was covering up the sensor with my hand, I had the theory that the sensor might be registering the capacitive coupling between the gold plate and my hand, but covering the sensor with a non-conductive material (a black anti-static bag) caused the same effect.

Thus the gold plate on the face of the Spreeta is thin enough to pass some ambient light that the photo-diode array can sense. So discoveries made during the verification phase include:

a) the LED can burn out with hard connection to the voltage rail
b) a potentiometer of 10K yields sufficient sensitivity to control the LED power
c) the Spreeta is sensitive to ambient light and all Assays should be run in a dark environment

We accomplished the final testing of the sensor hardware and software by dumping water on the surface of the Spreeta device and adding a grain of salt. I was very pleased to see the sensitivity of the Spreeta sensor in differential mode, but surprised by the shape of the absolute waveform. I expected to see a “textbook” curve where the waveform was flat across the top, then had a dip at the angle of Plasmon excitation. Instead, the waveform looks anything but smooth (see Figure 25).

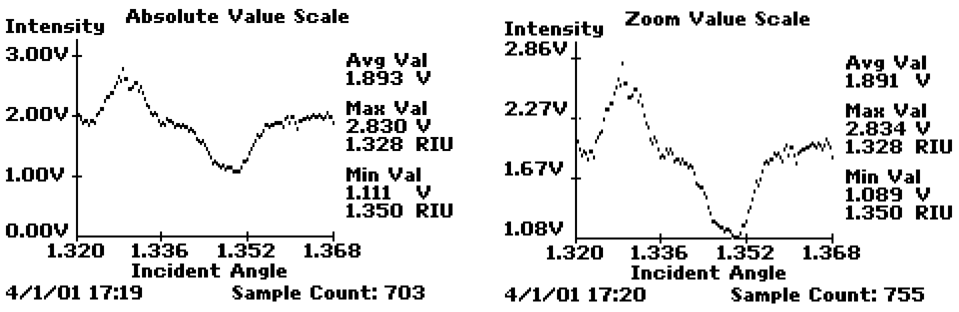


Figure 25. Initial Waveforms from Spreeta as Displayed on Accompli A009 Screen

7.3 DEVICE VARIATIONS

The next step in basic system verification was to determine if different Spreeta devices behaved in different ways. A “broad brush” technique was to purchase a standard tray of 25 devices, label each device and cycle all 25 through the sensor with deionized water applied to the surface to get a rough idea of the variation in the SPR waveforms. This proved very educational and of the 25 units purchased, we identified 4 units that showed the best waveforms in terms of flat lines with sharp dips at the resonant frequency: unit 10; unit 17; unit 20 and unit 22. Conversely, there was one unit, unit 19, which looked for all the world like a constellation of stars instead of an SPR waveform with a characteristic resonance dip.

Most of the waveforms presented in this thesis were generated on the 4 units that looked most like “text book” SPR curves. Note that this characterization of a tray of 25 units purchased and presumably fully qualified by Texas Instruments led to two conclusions:

1) there is a wide variation of SPR curve between parts
2) The Texas Instruments qualification process on their Spreeta devices is not based on “raw” waveform

Texas Instruments does vend a hardware interface kit and processing software for use with the Spreeta device. Based on the experience I have had with that software, it does extensive wave shaping on the raw data coming from the device and therefore may not be as sensitive to the general waveform as the design in my sensor design. On the other hand, there is something to be said for presenting the “raw” data because a heavy algorithm might occlude some detail of the SPR shift that proves critical to analysis. In a conversation with the Spreeta physicist, I discussed a phenomenon I had observed with the waveform changing its shape as well as shifting position and he had never been able to observe that type of behavior from the vantage point of the software processing in the Texas Instrument software.

7.4 FREE CHARGE

A repeating theme throughout our analysis of SPR and throughout this thesis is the concept of free charge and its affects on permittivity. The formal definitions for permittivity are presented in section 8.7 “FREE CHARGE (pH)” but as a overview, the basic premise is the permittivity of a material can be quantified by its capability to conduct electrical current. For deionized water, the capability to conduct electricity is low due to water's basic dipole arrangements causing relatively little free charge for conduction. However, by adding salt to water (e.g. adding free ions), the capability of that new solution to conduct electricity is greatly increased.

One of the earliest observations of this research is the difference between QCM and SPR: QCM measurements are based on mass loading whereas SPR measurements are based on changes in free charge. As illustrated in Figure 26, there is a significant charge difference between FITC (or urinine) and Alexa. Note specifically the purple charges on the triplet of aromatic rings at the base of the molecule in Alexa: 2x of SO_3^- and 1x of NH_2^+ which for the purposes of charge is a total of 3 (note that both positive and negative charge contribute to changes in permittivity). As an aside, the binding pocket in anti-FITC is believed to be at the base of the triplet and the charged ions at the base of Alexa which are absent in both FITC and urinine, are the cause for a molecule of the same size and structure to bind with much less affinity. Finally, note that the mass of FITC and Alexa are comparable (QCM "friendly") but the charge is significantly different (SPR "unfriendly").

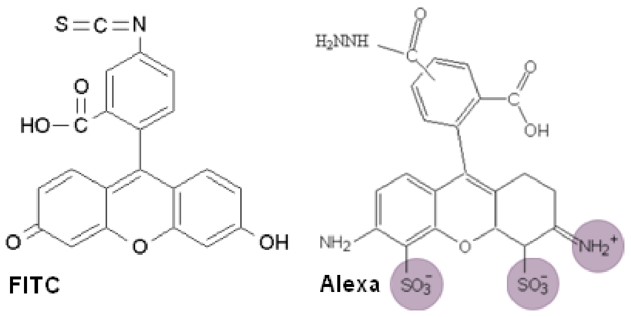


Figure 26. FITC and Alexa with Highlighted Ionic Charges

The assay used in this research was based on a positive tests for binding between anti-FITC isolated on the surface of the sensor (QCM or SPR) and measuring an increase in molecular binding. For positive tests, the sensor is baselined with no analyte present, then capture a constant measurement while introducing the urinine and use the change in the measurements as the “signature” for the binding events. For the negative tests, we use baseline as usual but introduce Alexa, which has the same basic structure and mass but weaker binding affinity. The results from these two assays are contrasted to eliminate any common sensor reactions (e.g. sensitivity to pressure changes due to the introduction of analyte) and the resultant waveform is the unique, positive signature. The four assays that can be derived from these components are shown in Table 2.

Table 2. Test Coverage for Sensor Characterization

Test Condition	Test Coverage
Uranine with no monolayer	Negative (Control)
Alexa with no monolayer	Negative (Control)
Uranine with anti-FITC monolayer	Positive
Alexa with anti-FITC monolayer	Negative

Once the system was calibrated and verified functional, the next step was to see how it performed in an actual assay with antibodies isolated on the surface. To run this assay, I had to enlist the help of a molecular research specialist. Given the portability of the sensor and supporting system, it was no problem at all to take the entire system into the biology lab to run the assay.

CHAPTER 8: IMMUNOASSAY ANALYSIS

Once the design of all phases (Hardware, Software and Monolayer) was completed, the next phase was characterization of system performance. We preformed the earliest tests by applying the building blocks directly to the gold surface using a micropipette as shown in Figure 27.

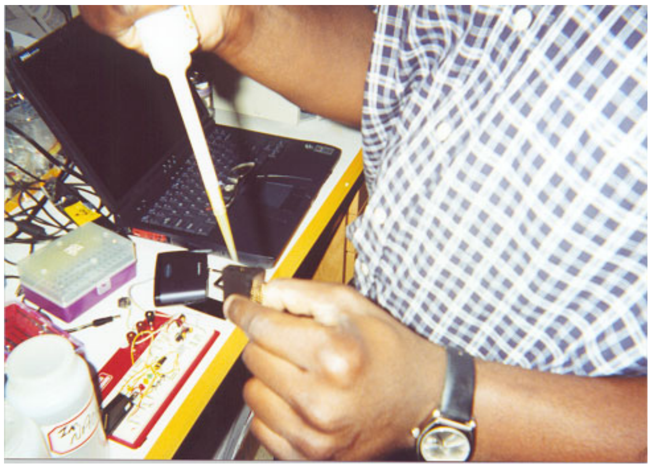


Figure 27. Micropipette Monolayer Building Blocks Directly Onto Surface

The earliest results were fairly positive as shown in Figure 28 but further analysis of these early results showed that molecular binding events were not the cause of the observed shifts. Although it was most gratifying to see such results in the earliest testing, there are unfortunately a myriad of side effects that must be carefully controlled, accounted for and/or eliminated if Surface Plasmon Resonance is to yield quantifiable results on molecular binding events. Note that Figure 28 was generated on earlier versions of the software and the red difference line has not yet been implemented in the Zoom and Absolute screen shots.

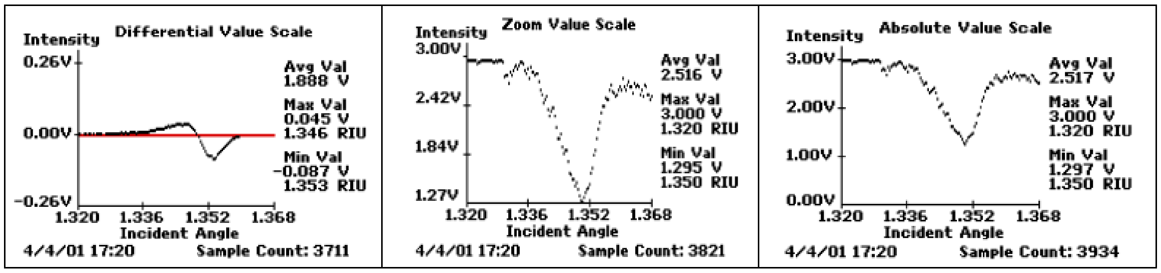


Figure 28. First Actual Assay Results

The remainder of this section will present the factors encountered while working to quantify the side effects that cause shifts in the waveforms. Some of the side effects cause changes in permittivity that are not the result of molecular binding events and others are artifacts of SPR in general and some are specific to SPR as implemented in the Spreeta device.

8.1 VARIATIONS/CALIBRATIONS BETWEEN DEVICES

As illustrated in Figure 29, the waveforms produced by different Spreeta devices vary significantly. This is one of the reasons the Spreeta device will not be capable of yielding absolute results in terms of SPR shift. The amount of shift registered is necessarily dependent on the initial waveform. It is possible to calibrate a single device and therefore measure the relative shift, but with the observed variation in waveforms a molecular event may well be occurring at a point of less sensitivity on one device than that same event on a device with more sensitivity at that frequency.

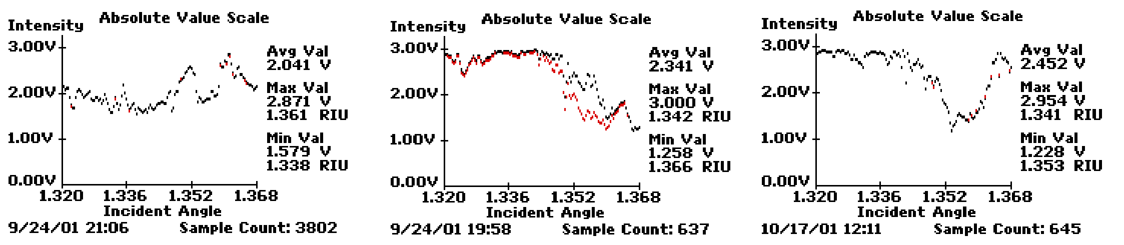


Figure 29. Device Differences in Absolute Waveforms

For example, the waveform at the far left of Figure 29 does not have the characteristic sharp dip at the resonant frequency. The waveform in the center shows a shift with more change in the "Y" direction than in the "X" direction. It is easy to imagine that equal changes in permittivity on the three different parts shown will result in different amounts of shift being registered by the biosensor.

In the comparison of SPR to QCM we looked at the possibility of using two Spreetas to cancel each other out thereby increasing the accuracy of our results. One issue with the Spreeta is wide variations from device to device that not only react to changes in permeability by shifting the SPR curve, but also deform the waveform differently from device to device. We explored the possibility of "differencing" two Spreetas as a baseline but we couldn't come to agreement on what to do when they registered differing results for the same input.

We then proposed the possibility of a threshold beyond which the parts were declared "too different" and the results ignored. We all agreed that QCM has fewer sources of variation and are thereby easier to calibrate against each other. In fact, we sometimes adjust a crystal to get it into a known starting frequency to achieve the very goal of calibrated (or at leas correlated) starting points. Calibrating a Spreeta to shift the SPR curve by a known quantity for a known input is certainly possible, but calibrating two to shift/deform prohibitively is problematic.

There is a possibility via software to measure the shift/deformation and extract the known differences between the two. We discussed this but it would require a lot of up-front calibration and correlation to a known good (or even theoretical "ideal") reference.

Sources of variation between Spreeta devices:

1) gold film thickness
2) optical properties of cavity “filler”
3) miss-match between top “filler” and bottom “filler”
4) alignment of CDC array (detector)
5) consistent sensitivity of each CDC cell in the array
6) variations in LED polarizer (rotation, tilt, etch quality, etc.)
7) variations in LED spectrum

All of these possible sources of variation are potential contributors to the Spreeta’s observed variations from part to part. Regardless or which ones dominate or which ones aren’t significant factors at all, there is a large observed variation in Spreeta devices. This variation makes it impractical to try to combine two Spreeta devices in some sort of negative feedback or interferometer configuration because their sensitivities are not constant across every frequency of the spectrum.

8.2 DIRECT LIGHT PENETRATION

Another property of the Spreeta, but not necessarily of SPR in general is the fact that the gold film on the face of the Spreeta is so thin (500 angstroms according to the TI engineer supporting our questions) that light can directly traverse it. Thus, as illustrated in Figure 30, the photodiode array measures all light hitting its surface and not just the light generated by the internal LED and reflected off the sensor surface. Both the light passing through the gold surface and the optical properties of the liquid also directly affects the photodiode array; liquid properties such as:

- Thickness
- Translucence
- Surface Tension

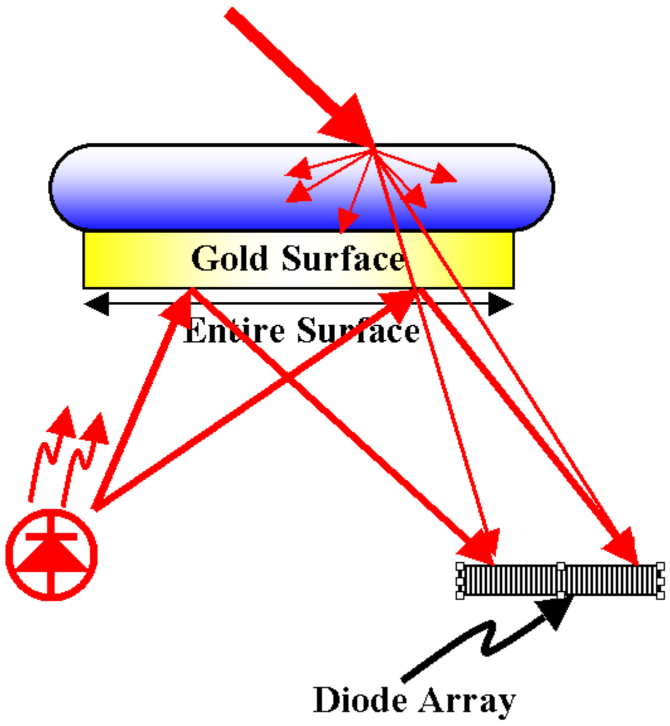


Figure 30. Illustration of Direct Light Penetration on SPR

8.2.1 COMPENSATING FOR DIRECT LIGHT PENETRATION

All of these properties can affect the travel of light including reflection, refraction and diffusion. Whatever light penetrates through the liquid and the gold film will be collected in the photodiode array and registered as signal. Ideally this would be a simple “bias” that could be subtracted from the desired signal by exploiting the integration period adjustment provided in the software. However, given the multiple non-homogenous optical layers in the stack-up of “liquid-on-sensor-surface” there’s no reason to expect all light penetration to be constant across the entire face of the photodiode array.

It would be necessary to measure the amount of interference per frequency and adjust each individual photocell, which isn’t a capability of the hardware but could theoretically be done in software with a “dark” vs. “light” calibration step. Of course, if this calibration were to be made, it would build the assumption into an assay that the amount of ambient light penetrating each cell would be a constant which implies the binding events would cause no shift in optical properties, the ambient light wouldn’t change (e.g. sun movement) etc.

8.2.2 ELIMINATING DIRECT LIGHT PENETRATION

The microelectronics laboratory is equipped with a high-quality evaporator so with the working hypothesis that since the gold on the surface of the Spreeta is only 500 angstroms thick, perhaps an additional 2000 angstroms on the surface would make the gold opaque. The extra gold, 2000 angstroms, was successfully added (high praise to Sang on his inventing a fixture for the Spreeta and his operation of the equipment) but unfortunately the resonant dip in the SPR curve disappeared.

The result was a straight line across the entire array indicating a constant value of light across all elements in the photodiode array. The device appeared to be functional as proven by adjusting the integration period and verifying that the waveform adjusted “up” and “down” indicating that the infrared LED was turned on and the photodiode array was measuring the light. Additionally, ambient light no longer penetrated into the photodiode array as verified by holding the device up to light and seeing no change in the waveform as measured by the photocell array.

The cause the disappearance of the resonant frequency dip in the sensor must be a result of a change in permittivity. A previous observation that the photodiode array has been placed such that it registers only liquid-based resonant frequencies is the clue: the thicker gold has caused a shift in the resonant frequency. Consider a gold slab thicker than the interrogation field of visible light (e.g. thicker than Infrared's frequency of 840nm). Then the only material interrogated by the light will be gold and the only SPR that could be registered is that of gold's. By thinning this theoretical gold slab into the interrogation field of the light, the resonant frequency becomes a combination of the gold and the material above it.

8.2.3 PERMITTIVITY BIAS VIA GOLD THCKNESS ADJUSTMENT

In the case of this experiment, the change in the gold's thickness from 500 angstroms to 2500 angstroms still leaves plenty of interrogation height into the material on the opposite side of the gold (even blue light is 400nm and the gold is only 250nm thick). However, the amount of gold interrogated increased 5 fold, which was enough to shift the resonance frequency to one beyond the capability of the photodiode array to measure. This is merely a mechanical limitation of the Spreeta device, but a real one nonetheless. Conversely, thickening the gold on the surface of the sensor may be a good method of biasing the device to read permittivity on materials too high to register on the Spreeta's photodiode array.

8.3 TILTING

As state above, early testing was with a Spreeta attached directly to the breadboard as shown in Figure 1. In this configuration, the slope of the gold surface is very sharp and it's not possible for liquid with any viscosity to sit still on such a steep incline. Thus a simple solution to this problem was to tilt the whole breadboard until the board was at a steep angle and the gold surface was horizontal. By tilting the board, the chemicals, proteins and antibodies were applied directly to the gold surface via micropipette as described in CHAPTER 4: "MONOLAYER (BIOLGICAL INTERFACE) DESIGN".

With the monolayer chemicals thus introduced to the surface, the procedure called for introduction of the analyte and the real-time recording of the analyte binding events. The time allotted for the binding was on the order of 10's of minutes and during this time, the SPR curve did indeed shift which was at first a very exciting observation. However, the experiments were never repeatable so it became a source of frustration as to why the binding events sometimes showed shifts to the left, other times shifts to the right and most surprisingly, shifts that changed direction.

8.3.1 OBSERVATION

The important discovery during this analysis phase was that the Spreeta device is sensitive to "tipping". As illustrated in Figure 31, the Spreeta is very sensitive to the actual depth of the liquid on the face of the sensor. The observed phenomenon was the SPR curve would shift when the Spreeta was tilted and it was actually possible to watch the SPR curve "crawl" off the end of the diode array. Needless to say this discovery was both a relief to know another source of variability in SPR technology along with another puzzle as to why the Spreeta is so sensitive to tilting.

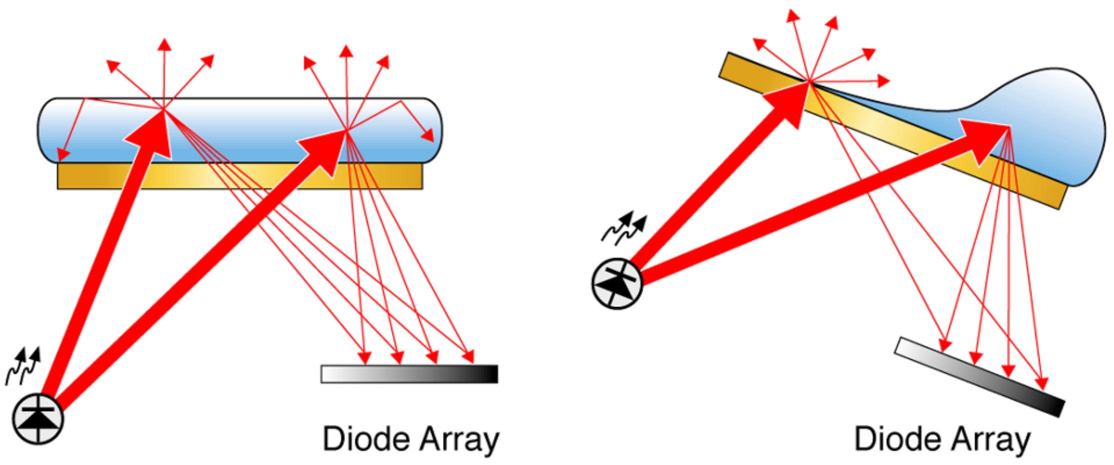


Figure 31. Illustration of Tilting Effects on SPR

8.3.2 THEORETICAL EXPLANATION

This phenomenon was not easily dismissed as the surface going “dry” because at the molecular level, and even at the wavelength interrogation level the surface certainly didn’t dry out. So the theory of the observed shift has two components: direct light penetration and change in permittivity. The direct light penetration theory has to do with the changing thickness of the liquid and the light source from inside the Spreeta penetrating the gold surface and being reflected back down into the diode array in some fashion. In this theoretical model, the light reflected would be affected by changes in the liquid depth, surface tension and the relative density of impurities. However, while reflected light is a possible source of “noise” to the diode array, it is not the cause of resonance or of resonance shift in SPR technology. A change in the amount of reflected light would tend to shift the entire waveform up and down as the amount of collected light might vary, but it would not easily explain a sift in resonant frequency.

The second component of the observed change in waveform goes straight back to the basic premise of Surface Plasmon Resonance: a change in permittivity. If shifting (pouring) the liquid off the surface of the Spreeta changes the permittivity at the gold surface, the SPR curve would shift as well. Since the permittivity measured by the Spreeta is the concentration of free ions at the surface, it is reasonable to conclude that the concentration of free ions is changing as the liquid is poured away. This would imply that either the free ions (e.g. analyte, Protein “A”, antibodies and Protein “A”/Antibody complexes) are flowing away with the liquid or are being left behind as the fluid leaves.

Either condition is possible but there’s another source of impurity not associated with the buffer or the biological particles: the buffer had been mixed with a 1% solution of agarose gel. This gel has the property of building up a meshed structure throughout the body of the liquid it inhabits [20]. Therefore it is unlikely to be as viscous as the liquid flowing through it. Thus the theory is the agarose gel forms a structure throughout the liquid and when the liquid is pulled away, in this case by gravity, the agarose remains behind. Thus the agarose gel being left behind while the liquid squeezed out from around it could explain the change in permittivity. The permittivity change is the increase in concentration of the agarose gel as the TAE buffer concentration decreases. This is similar to the evaporation phenomenon discussed in section 8.4 “EVAPORATION” in which all impurities would increase in concentration as the liquid evaporates away.

8.3.3 MAINTAINING HORIZONTAL POSITIOINING

The obvious fix to this issue of tilting is to attempt to keep the surface horizontal for the duration of the molecular binding time. However, this proved impractical due to the instability of the circuit board sitting at a sharp angle on a bench top. Simple acts like placing the black bag over the board to avoid ambient light was enough to change the angle of the board and introduce a “tilt shift”. Even taping the board to the bench top was not enough to keep the system stable. The challenge is to maintain a constant concentration of free ions and the tilting phenomenon had to be eliminated.

8.3.4 SUMERGE SENSOR SURFACE

This adjustment was based on the theory that if the Spreeta is sensitive to liquid thickness, and more importantly changes in liquid thickness, then it might make sense to eliminate any possibility for changes in liquid thickness. To accomplish this “infinite” depth of liquid, the Spreeta was partially submerged in a beaker filled with analyte. The goal is to completely submerge the gold surface without submerging the pins, as the pins carry electrical signals and a beaker of analyte will short out the pins and make the device inoperable (or potentially damage either the driver or the Spreeta).

This experiment required a change in the circuitry: specifically, the Spreeta had to be put on a long tether (aka. umbilical) so the driver circuitry could remain on the bench top while the Spreeta was free to be dipped into the beaker. The cable was hand-wired and roughly 2 feet long with the ground line wrapped around the signal pins for increased noise immunity.

8.3.4.1 EXPERIMENTAL RESULTS

The results of the Spreeta on a tethered cable dipped into a beaker of analyte were encouraging in that the set-up remained fully functional and the Spreeta generated a consistent reading of a characteristic plasmon resonance dip, but the experiment became too unconstrained. Some of the difficulties using the device in this configuration were:

1) liquid getting on the pins of the device (e.g. the Spreeta dipping too far into the beaker)
2) motion of the Spreeta whenever the beaker was even slightly disturbed
3) ambient light (we even unscrewed the light bulbs in the lab)
4) introducing the analyte: bubbling vapor into the beaker was especially challenging

8.3.4.2 APPLICABLE USE CASES

This experiment showed some promise for a specially adapted environment to this arrangement in which the Spreeta pins are sufficiently protected from the analyte. Another challenge with this arrangement is the required bulk of the analyte: enough to dip the Spreeta. The use case for "dipping" might be a food processing plant in which the analyte is constantly flowing across the stationary Spreeta device. In this use case, there would be plenty of depth for the Spreeta, the environment could be ambient controlled and the sensing to be a change in some characteristic of the product beyond manufacturing tolerance. For example, a yogurt manufacturer where the product has become too alkaline (sour) or a fish hatchery that requires a carefully controlled balance of nutrients in the water. The Spreeta could simultaneously monitor for ambient light, which might be an indicator that the water level in the tank has become too low.

8.4 EVAPORATION

Part of the initial design of the Spreeta monolayer was to cover over the monolayer with a 1% concentration of agarose in the TAE buffer to build a "wet" environment for the antibodies. This type of covering was theorized to keep the antibodies healthy and active for a prolonged period of time. Allowing the antibodies to dry out would render them incapable of performing their analyte attachment activities. Further, the analyte was theorized to be small enough to work its way into the hydrogel and attach to the antibody.

8.4.1 Observation

Our findings show that during an assay the hydrogel evaporates off the face of the surface in about an hour. This observation was very surprising and we wondered why the evaporation was so rapid. We expected the hydrogel to survive on the surface for days as opposed to minutes. Further, what we thought was antibody activity as registered by a shift in the SPR waveform was merely the observation of the evaporation of the liquid on the surface.

8.4.2 THEORY

Our theory as to why evaporation would cause a shift in SPR is a direct response to a measured changed in permittivity. In this case, the permittivity of the biomaterial in the interrogation field of the Spreeta device would be an increased concentration of “impurities” left behind by the pure water transitioning into vapor (see Figure 32).

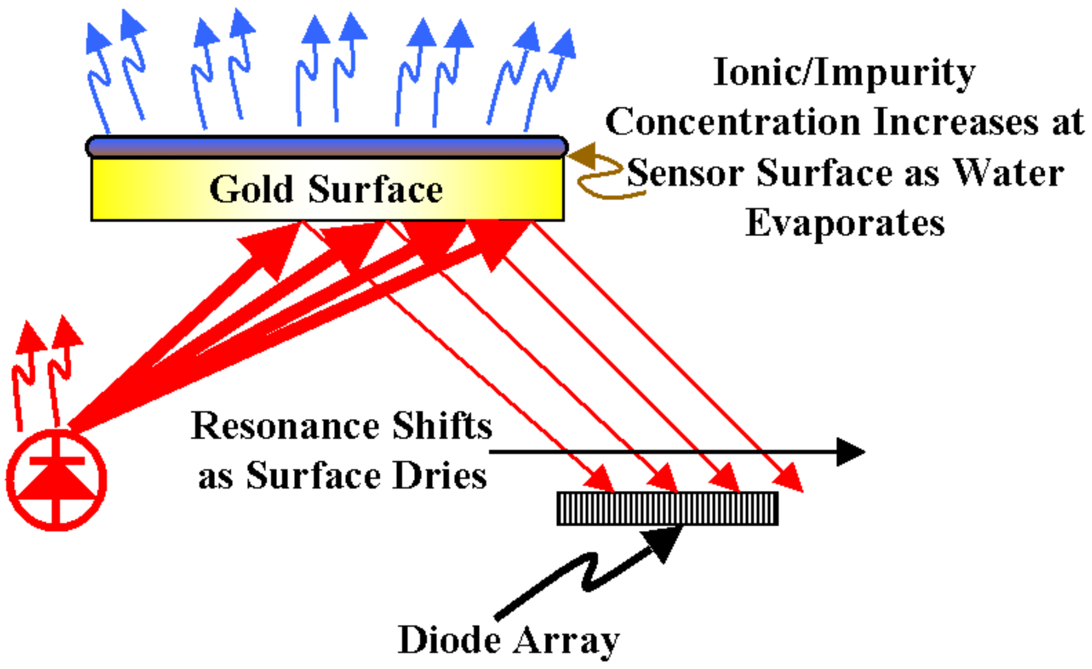

Ionic/Impurity Concentration Increases at Sensor Surface as Water Evaporates
Gold Surface
Resonance Shifts as Surface Dries
Diode Array

Figure 32. Illustration of Dry Surface Effects on Spreeta

Thus to keep the hydrogel on the surface longer we made three adjustments:

1) increase the concentration of agarose in the hydrogel
2) control the ambient temperature
3) reduce the power of the internal Spreeta LED

Increasing the concentration of the agarose in the hydrogel should have significantly slowed the evaporation process. Additionally, keeping the hydrogel in a cool environment should likewise slow evaporation as before we had been quite cavalier about working in Atlanta's hot weather. Neither experiment showed a significant decrease in the evaporation rate. So we turned our attention to the gold surface of the Spreeta, which perhaps was acting like a hot plate.

8.4.3 MITIGATION STRATEGIES

We reduced the power of the Spreeta's internal LED by adding circuitry and software to turn off the LED when we were not capturing data and turning it on only when running an interrogation of the surface. We also adjusted the light intensity. Neither adjustment had any significant affect on the evaporation rate of the hydrogel.

As a final experiment, we covered a Spreeta surface with hydrogel and without even turning on the device, placed it in a 4^{o}C temperature chamber. In less than 24 hours, the surface was dry. Thus we abandoned the idea of controlling evaporation on the surface of a Spreeta open to ambient air. The best environment for Spreeta usage would be one where evaporation was eliminated and a constant supply of analyte was being presented to the Spreeta surface.

8.5 FLOW CELL

Many of the issues discovered with this research can be mitigated using the "Flow Cell" available directly from Texas Instruments. This cell is a custom fit mechanical housing for the Spreeta device and exposes a sealed liquid channel across the gold surface of the inserted device (see Figure 33).

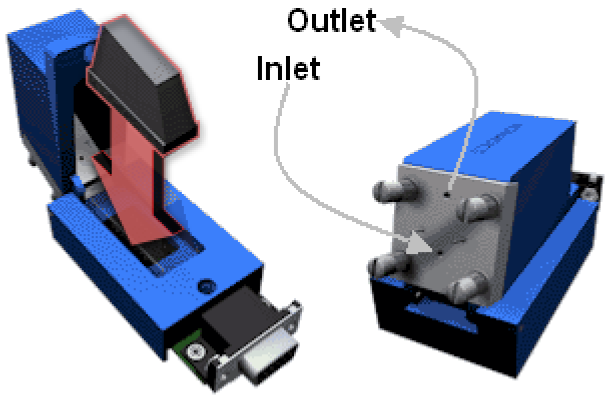


Figure 33. Spreeta Flow Cell

8.5.1 FLOW CELL ADVANTAGES

The flow cell is designed to mitigate some of the factors that affect SPR readings such as:

1) elimination of ambient light
2) constant liquid thickness

8.5.2 FLOW CELL USAGE

Additionally, the flow cell provides a convenient mechanism to introduce analyte to the Spreeta by using a syringe to add the solution containing the analyte. We used this tool to run both positive (urinine) and negative (Alexa) tests on the Spreeta device. Some of the issues to overcome in using the flow cell are:

1) Keep a constant pressure
   - don't stomp on the syringe
   - block outlet when changing syringes
2) Avoid air pockets (don't let the paper towel at the outlet suck out the liquid)
3) Fill the entire chamber when introducing new liquids

By following these simple rules, we were able to achieve results that were more consistent with our expectations. There were no more issues with trying to perform assays with the lab lights turned off or covering the Spreeta with a baggie. The flow cell does, however, introduce a phenomenon with building the monolayer: with the Spreeta in the flow cell, the channel for protein "A" complex binding is now 3-dimensional.

8.5.3 CONCERNS WITH FLOW CELL USAGE

The channel is narrower than the full size of the Spreeta gold surface so rather than build the monolayer on the gold surface with a micropipette, we built the monolayer by introducing the chemicals and biomaterials via the syringe. Thus we were always concerned about the cover/lid of the flow cell and the gasket that sealed the lid to the gold surface, wondering if they were introducing contamination or if the protein “A” or the anti-FITC would stick to those materials. Additionally, we were concerned about turbulence in the channel during introduction of the analyte.

8.6 MULTIPLE SIMULTANEOUSLY INDUCED PLASMONS

At this next step we gained valuable insight into the Spreeta itself: rather than think of the Spreeta as a single SPR biosensor, it is more conducive to proper interpretation of the output to think of it as 128 simultaneous SPR engines all operating at progressively increasing wavelengths. This must be clearly understood because the temptation is to assume the biolayer is “text-book perfect” across the entire face of the Spreeta sensor surface and there should be exactly one resonant frequency at all times. Our findings demonstrate that this is assumption will lead to false interpretation of the output.

As illustrated in Figure 34, it is straightforward to imagine several locations along the face of the gold film resonating with the inconsistent levels of permittivity within the interrogation fields. Three possible causes for simultaneous resonance are illustrated: resonance at the expected wavelength induced by the ideal conditions (far left), resonance at the wavelength interrogating a patch of non-specific (random) binding, and resonance due to variations in interrogation depths caused by shorter wavelengths (far right). For the third example, consider a charge distribution gradient that is highest near the structure of the monolayer and gradually lessening as distance increases into the random particle distribution of the buffer. In such a gradient the “blue shifted” interrogation will be stronger in the higher charge densities but the “red shifted” interrogation will penetrate farther into the lower charge densities.

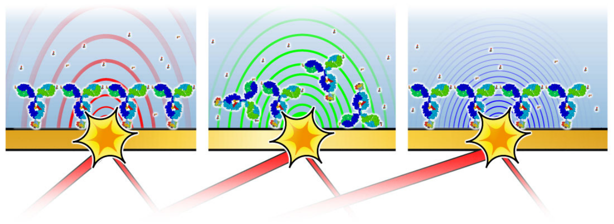


Figure 34. Multiple Simultaneously Induced Plasmons

Our observations of the waveform output from the Spreeta correlate with this “multiple simultaneous resonance” theory in that we have clearly seen cases where the curve widened or narrowed as opposed to strictly shifting right or left, indicative of a change in resonance in one region without a complimentary change at the opposing region. Thus it is beneficial to study the “raw” data coming from the Spreeta, which may be yielding quantitative information about the monolayer itself as opposed to just operational information about resonance shift. It may be yielding information on both shifting and “skipping” at the same time.

8.7 FREE CHARGE (pH)

The underlying premise of Surface Plasmon Resonance is its measurement of Relative Permittivity. The Relative Permittivity can also be defined as a material’s “dielectric constant”:

A dielectric material is a substance that is a poor conductor of electricity, but an efficient supporter of electrostatic fields. If the flow of current between opposite electric charge poles is kept to a minimum while the electrostatic lines of flux are not impeded or interrupted, an electrostatic field can store energy. [17]

In layman's terms, permittivity is the measure of how well a material can conduct electricity. It is inversely proportional to a material's dielectric properties which is a measure of the extent a material will hold it's charge. A good dielectric is a good insulator; a material with high permittivity is a good conductor. The amount of free charge, or charged particles, in a liquid determines its conductivity or permittivity. Pure water (de-ionized water) is the standard of neutrality for pH (pH=7) but this doesn't imply that there is no free charge in deionized water. In fact, there is a balance of free positive (H^+) and free negative (OH^-) particles.

However, this research supports the observation that free charge, as measured by pH, can also be directly measured by the Spreeta device and therefore is a measure of the permittivity registered by the sensor. Permittivity has a real component and a complex component. The real component represents the arrangement of the atoms in a fixed structure. An example is the dielectric material used in a standard capacitor. In this example,the dialcomplex part of permittiity is ignored. However the complex part of the permittivity is due to mobile ions. So although a capacitor dielectric has negligible free ions, a biological immunoassay in aqueous solution does not. Thus one would expect a profound effect on the optics of a SPR biosensor due to the free ions within the field of interrogation of the incident light excitation.

8.7.1 MEASUREMENT PROCEDURE

As the foundation of the biological interface is the TAE buffer, the procedure to measure the effects of free charge (pH) on the Spreeta surface was simply to eliminate all other biological materials and work only with the buffer. Thus a variety of different pH buffers were prepared using TAE as a base and mixing in appropriate concentrations of either Amino Ethanol ($NH_2CH_2CH_2OH$- Base) to lower the pH of the solution or Sulfuric Acid (H_2SO_4 -Acid) to raise the pH of the solution. With these mixtures, it was possible to adjust only one variable of pH while keeping constant the other characteristics of the TAE buffer. The final pH of each solution was verified using a pH meter in the biology lab.

Table 3. pH of TAE Solutions

TAE_{802}	pH 8.02 (used as reference)
TAE_{868}: TAE + Base	pH 8.68
TAE_{717}: TAE + Acid	pH 7.17
TAE_{627}: TAE_{717} + Acid	pH 6.27
TAE_{417}: TAE_{627} + Acid	pH 4.17
Water	pH 5.76

As shown in Table 3, the pH of the TAE buffer was adjusted and measured. Additionally, the water in the Biolab was measured as well and found to be pH 5.76 (as opposed to 7.0). The de-ionized water is set to be pH of 5.76 due to its use in a biology lab. It is calibrated to be as close to physiological pH as possible to facilitate the bioresearch typically conducted in the lab.

8.7.2 THEORY OF EXPECTED RESULTS

SPR measures permittivity and as such measures the biolayer's capability to sustain an electrical field. The more free charge there is in a material the less that material can maintain an electric field. Thus at pH of 7.0, the TAE buffer should be the least ionic and have the lowest permittivity. Either raising the pH (increased positive free ions) or lowering the pH (increased negative free ions) will increase the permittivity and register a shift to the SPR curve.

As illustrated in Figure 35, as pH changes the SPR curve should theoretically shift accordingly. The SPR curve will be at the farthest point to the right at the most neutral pH and shift left for both positive and negative ionic concentration increases. Thus an increase in positive free charge: a change in pH from 7.0 ➔ 6.0; will shift the resonance curve to the left. An equal but opposite increase in negative free charge: a change in pH from 7.0 ➔ 8.0; will also shift the resonance curve to the left and by the same amount as the positive shift.

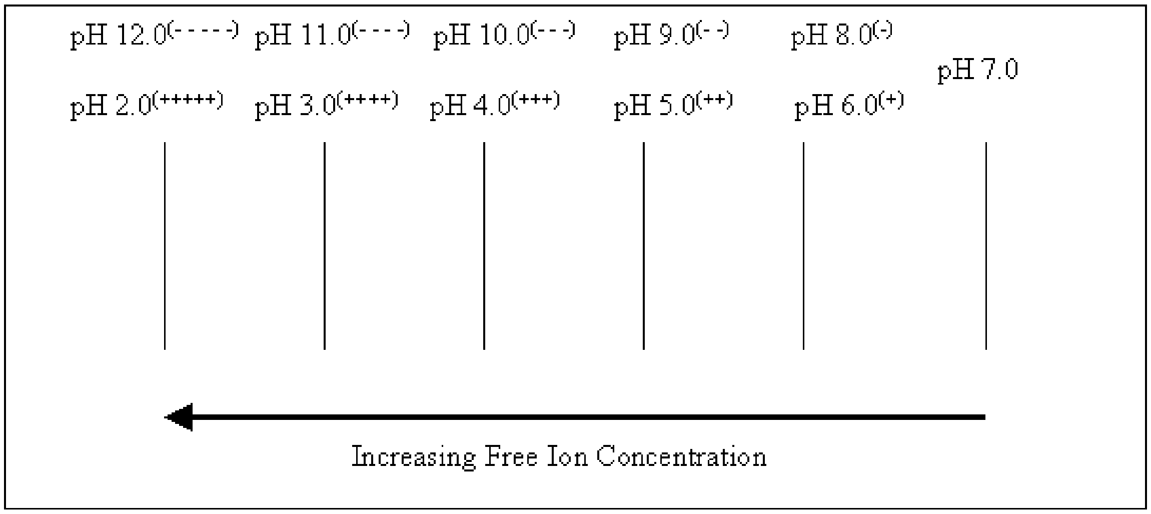


Figure 35. Relative SPR Curve Shift Directions Resulting from pH Changes

Thus the SPR curve can shift left or right as the concentration of free charge increases or decreases in the field of interrogation. It all depends on the free charge in the "before" case as opposed to the free charge in the "after" case.

8.7.3 MEASURED RESULTS

The first step was to capture the initial waveform at TAE buffer pH 8.02 as shown in Figure 36. With TAE_{802} as the baseline, the result of adding other solutions will cause a left shift in the SPR curve indicating "more ionic" and a right shift for "less ionic":

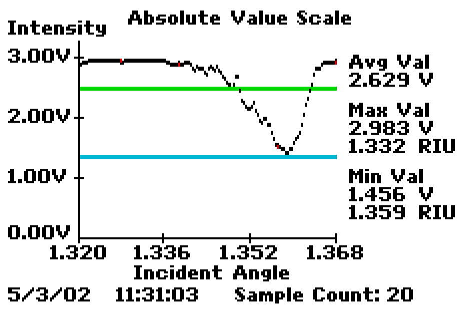


Figure 36. Baseline of TAE_{802} w/o pH Adjustment

Capture the initial waveform at TAE buffer pH 8.02. This will cause a left shift in the SPR curve indicating “more ionic” and a right shift for “less ionic”:

pH 7.0	➔ Right Shift 1 graticule
pH 6.0	➔ No Shift (same ion concentration as 8.0)
pH 5.0 & 9.0	➔ Left Shift 1 graticule
pH 4.0 & 10.0	➔ Left Shift 2 graticules

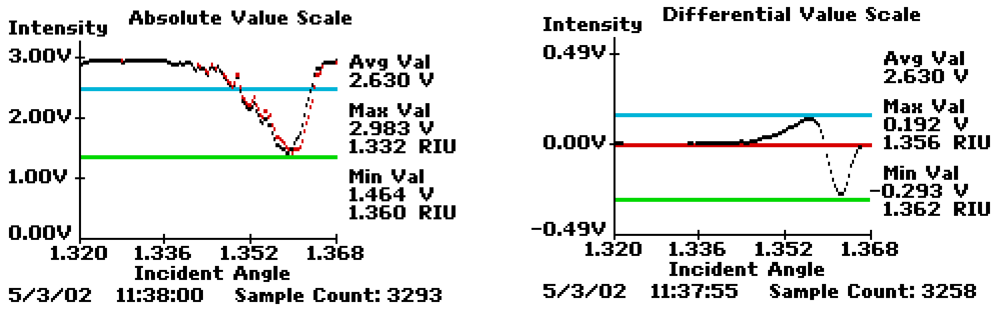


Figure 37. TAE_{802} ➔ Water: pH 5.76

As shown in Figure 37, the result was a very large shift to the right. We can’t quite explain the drastic difference or the direction of the shift since the expected result was a small shift to the left (5.0 would yield a 1-graticule shift left > 5.0 so the shift left should be even smaller than one graticule with 5.76). Also note the deformation of the curve in the absolute picture: the red (new position) curve doesn’t hit the green cursor as the black (original position) curve does. Note also the red curve is more “rounded” at the resonance curves.

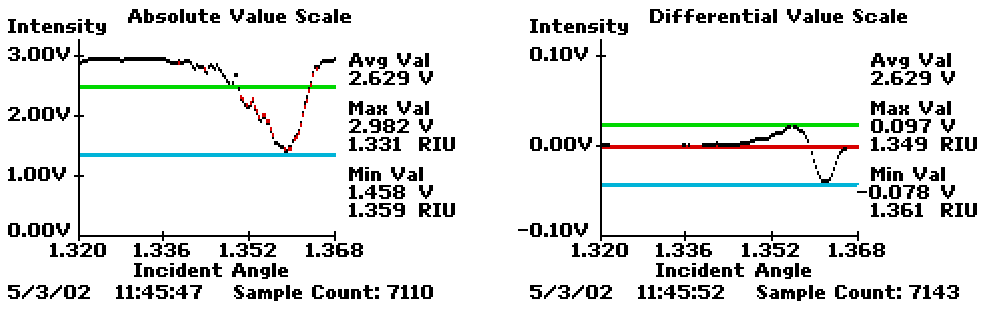


Figure 38. Water: pH 5.76 ➔ TAE_{802} (expect the shift to disappear)

As shown in Figure 38, the result was very close to "as expected" although not perfect. Possibly the water cause a permanent shift by changing the characteristics of the gold film surface. Perhaps the observation is due to a hydrophobic or hydrophilic change at the sensor's surface.

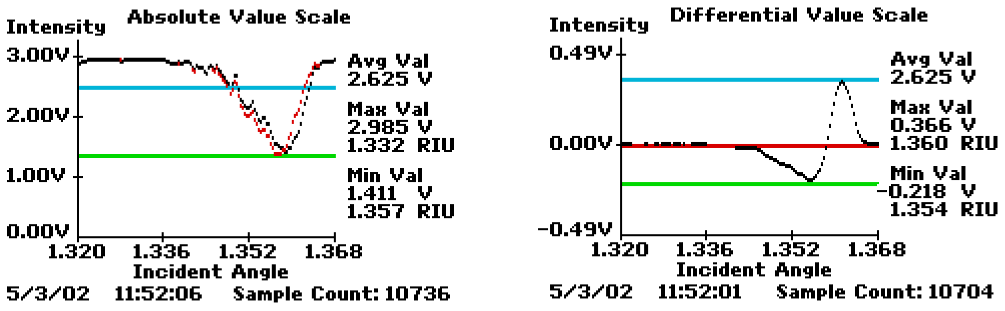


Figure 39. TAE pH 8.02 ➔ TAE_{868} (expect small left shift)

This result, as shown in Figure 39, shows the waveform is relatively close to "as expected" and the Spreeta is capable of measurable sensitivity to a relatively small change in pH:

8.68-8.02 = 0.66 ➔ 0.366V High/-0.218V Low Change

Thus a graticule of a full pH change would be expected to be even bigger than the one registered for this change.

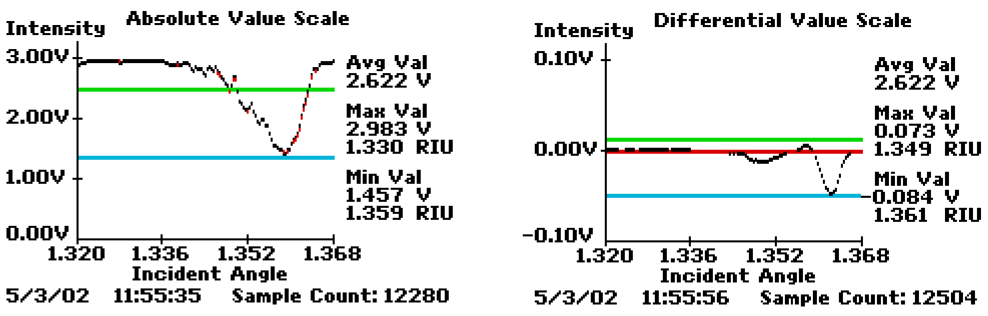


Figure 40. TAE + Base pH 8.68 ➔ TAE pH 8.02 (expect return to original)

As shown in Figure 40, the waveform is very close to original waveform except there's even more deformation of the original curve which can easily be seen on the differential plot. The waveform seems to have an overall widening towards the right as opposed to a shifting in either direction as can be seen from the multiple negative dips with no positive peaks.

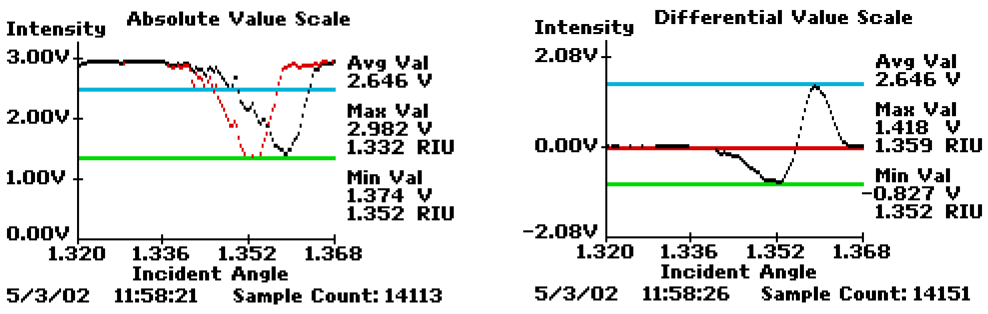


Figure 41. TAE_{802} ➔ TAE $_{717}$ (expect near maximal right shift)

The waveform in Figure 41 is the biggest shift left registered yet. This result is very hard to explain using only pH as a motivation for the measured change in permittivity, but it was clearly measured and preserved here for further analysis.

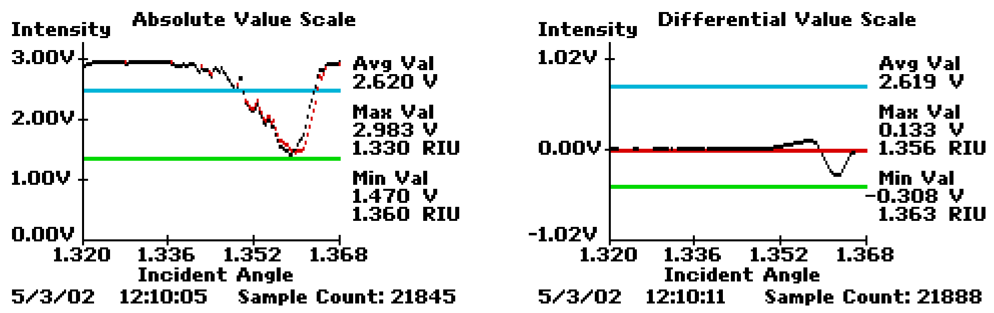


Figure 42. TAE$_{717}$ ➔ Water pH 5.76 (expect "Water" waveform as in Figure 37)

As shown in Figure 42, the original waveform is as expected; it is basically the water waveform with the characteristic rounded SPR curve except that it is shaped differently based on the differential data:

Figure 37 ➔ High: 0.192/Low: -0.293 vs. Figure 42➔ High: 0.133/Low –0.308

8.7.4 OBSERVATIONS

Unfortunately, these exact same results could not be obtained with multiple trials. There is obviously something affecting the SPR curve shift readings either over time or between assays. Under the assumption that the Spreeta measures only permittivity, there's no reason to think that "water early" is different than "water late". Variations over time could be:

- heat
- shifting inside flow cell (gasket, Spreeta, Inlet & Outlet tubes, etc.)

- leakage of liquid onto electronics
- air pockets trapped inside Spreeta (perhaps in the form of bubbles)

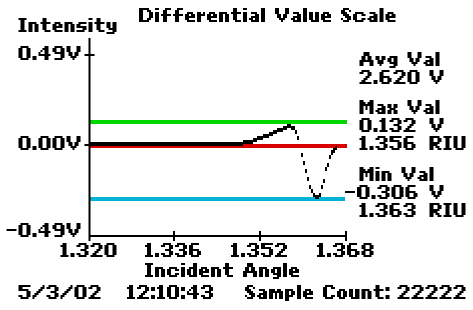


Figure 43. Water pH 5.76 ➔ TAE$_{627}$ (expect very little shift)

As shown in Figure 43, the final result was a large right shift.

8.7.5 NEXT STEPS:

These results need to be correlated with the QCM results (where no shift due to pH change is expected since changing pH shouldn't make a "stiffness" change to the quartz). We also need independent measurements of our prepared samples to validate these results. As it was a surprise that the water in the lab was so acidic, it may be that the TAE didn't start from pH 8.02 which would be an important factor to interpreting these results.

CHAPTER 9: CONCLUSIONS

This department has gained significant insight and even a publication as a direct result of this research. This project enjoyed early successes in the hardware design and the software design (both algorithmic and device control). Additionally, this research provided curtail insight for Surface Plasmon Technology as another alternative for detecting and monitoring molecular events in the sensors portfolio.

9.1 SUCCESSES

First of all, the most important conclusion is that the system is fully operational and can detect and display analyte attachment for immunoassays in real-time. The system is robust enough to withstand multiple runs of experiments and as portable as hoped. From running salt-water experiments in my office, to running immunoassays at Georgia Institute of Technology, the system performed without failure.

The goal of this experiment was to determine if a commercially available Surface Plasmon Resonance device is a viable component in a biosensor, and the result of this experiment shows that indeed, the Spreeta from Texas Instruments can be used quite easily and reliably to detect changes in liquid assays: from changes in bulk density (increase in salt content) to analyte attachment in an immunoassay, the Spreeta yields results.

The amount of hardware and software required to capture these results is quite minimal and definitely low cost: the hardware required is simple, industry standard and readily available, and the amount of software required to run the entire system (including the screen draw routines) is only 1555 lines of "C" code (including comments, and I write a **lot** of comments). The information to attach the device is right on the Texas Instruments website and the direct support from the help desk was easily accessible: all e-mail was answered in less than 24 hours.

9.2 OBSERVATIONS

To successfully apply the general technology of SPR and specifically with the Spreeta deice, there must be strict control on many, many parameters. Any change in any of the listed parameters will directly result in a shift of the resonant frequency and therefore register as a characteristic change. The amount of shift due to these artifacts can likely be measured and subtracted from the shift leaving just the shift due to the actual events under test. Thus this technology can be applied to molecular events but must be carefully controlled and characterized.

9.2.1 AMBIENT LIGHT

Although this is not a general artifact of SPR, the Spreeta's design, with the thin film exposed to light penetration, makes ambient light penetration of the film an artifact that will affect the waveform.

9.2.2 VISCOSITY/EVAPORATION

As the water content of the liquid under test evaporates, the "impurities" in that water will increase in concentration in the remaining liquid. This will necessarily increase the "impurities" in the field of interrogation and once again cause a shift in the SPR resonant frequency.

9.2.3 pH

A change in pH will directly affect the permittivity over the entire liquid field, which will directly cause a shift in the SPR resonant frequency.

9.2.4 TEMPERATURE

As temperature changes, a liquid typically becomes either more or less dense. A change in the density of the liquid will change the concentration of the “impurities” in that liquid and perhaps even the permittivity of the liquid medium itself. Either of these phenomenon will cause a shift in the SPR resonant frequency.

9.3 IMPROVEMENTS AND ENHANCEMENTS

Several improvements and enhancements can be made to the biosensor design as currently implemented for this thesis effort.

9.3.1 SOFTWAE ALGORITHM

1) Allow the vertical sensitivity of the differential mode to be dynamic: having the value fixed at 5.86mV may not apply to all assays.
2) Provide a horizontal zoom in addition to the vertical zoom; in fact, the vertical zoom has not proven very useful in all of the testing to date.
3) Print more of the significant digits for the RIU reading on the screen, since there’s only a 0.042 RIU spread between the minimum and maxim values, the display should provide more information on the exact location of the minimum & maximum
4) Provide two scroll bars the user can move over the waveforms with the readings along the side of the screen reporting the pixel value and RIU at each scroll bar

5) Add the threshold capability outlined above and enhance the UI so the user can identify what action to take when the differential screen crosses the thresholds (e.g. enter an e-mail address and text to be sent, enter a phone number to call and record a message to be played, etc

9.3.2 HARDWARE

1) The low-power mode of the hardware could be explored and perhaps turning off the LED can be compensated such that it doesn't affect the readings.
2) Eliminate the power-line hook-up and let the Accompli 009 source the power for the Spreeta and the connection circuitry.
3) Build a cable adapter between the Accompli 009 serial port and a PC in order to pump the data through to the PC-based algorithms developed by Texas Instruments. This would provide direct correlation between any results gathered by this sensor design and any published results using the standard software package.

9.4 APPLICATIONS

This biosensor can readily serve any application where the goal is to detect real-time drift away from a calibrated (e.g. "known good") starting point. An application in manufacturing process control springs immediately to mind. For example the sensor can be immersed *in vitro* into a chemical processing plant and upon detection of a drift outside the calibrated limits (or better yet drifting towards violation of the calibration limits) the biosensor can wirelessly notify the system administrator and/or adjust the process' inputs.

This sensor can continuously monitor and, as a component in a feedback control environment, maintain the quality of the output. There are practical applications in food processing, drug manufacturing, oil refineries and waste management. Note that this sensor is only suited to liquid applications as constrained by the Spreeta device and as implemented does not have direct application in any sort of “vapor phase” analysis.

APPENDIX A: HARDWARE SCHEMATIC DETAILS

As shown in Figure 44, the hardware schematic for this project was hand-drawn instead

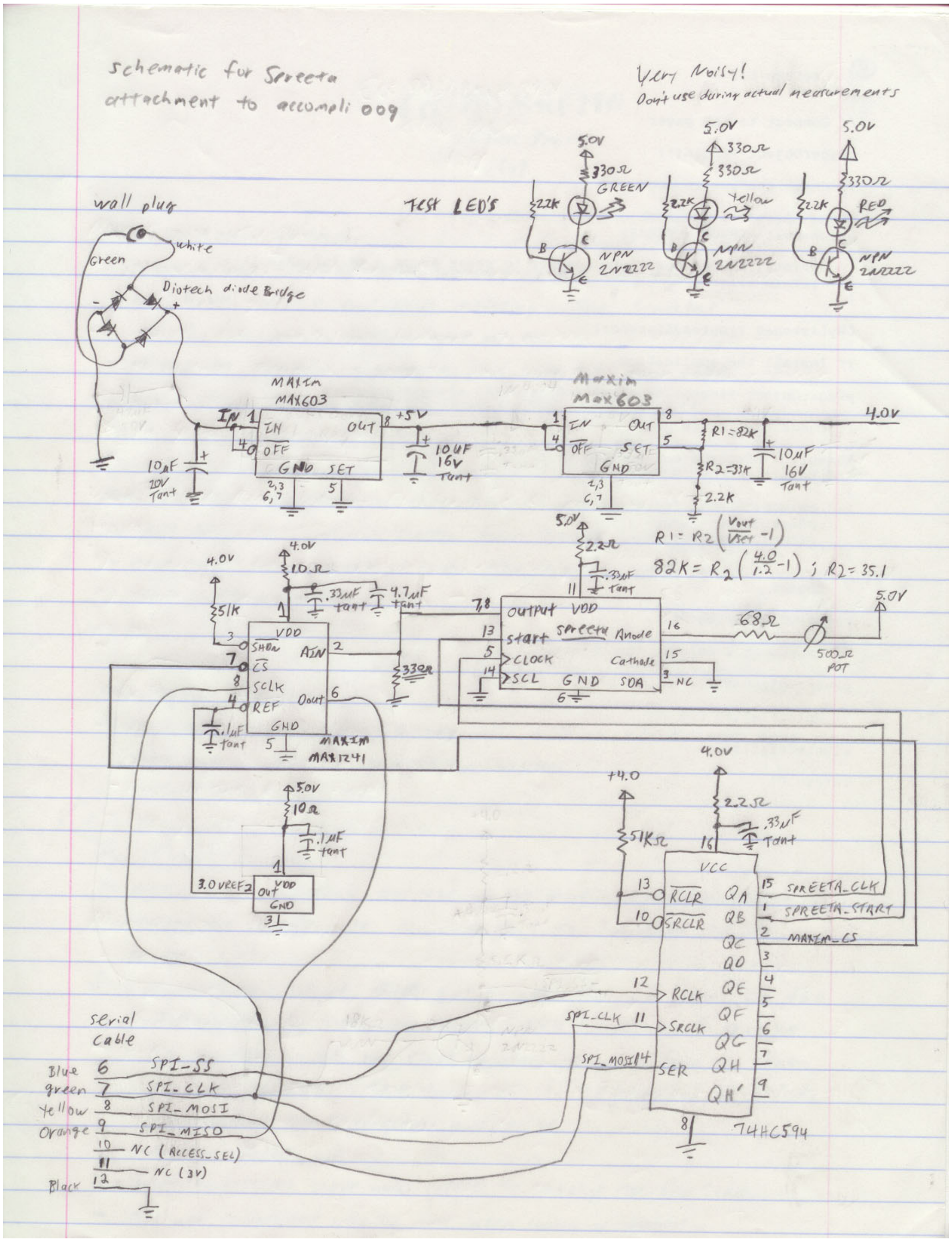


Figure 44. Hardware Schematic

of captured on a logic-capture system. Since the project was implemented on a bread-board with through-hole parts, there was no compelling reason to use a logic-capture system. This was the fastest way from design to implementation. As explained in CHAPTER 5:, “HARDWARE DESIGN”, the entire project was built from readily available componenets such as free samples from the vendors and low-cost components from Radio Shack^{TM}. The design was intended to be robust and never cause any loss of time which is why there are extra circuits like a diode bridge in the power input circuit to avoid reversed tip polarity.

A stack of 20-pin sockets was always used to ensure the integrity of the connectors to the Spreeta and for easy repair if a pin was broken during instertion/extraction. The connector to the Accompli 009 as well as the connector to the Spreeta was soldered and when verified was covered over with plastic cement (non-conductive) so no amont of yanking on the wires (which happens a lot when a board gets dragged aound for two years) would break the connections.

The power supply need only deliver a DC voltage of greater than 5V so any supply from 6V up will work well. The only constraint on the power supply comes from the 5V linear regulator: the board draws approximately 200mA when running full out and if the DC voltage is too high, the linear regulator will be converting a fair amount of that voltage drop into heat. There’s been no issue with heat using a 9V supply but if a 12V supply is readily available, there’s no reason to hesitate to use that power supply instead. Simply check the 5V regulator to make sure it isn’t overheating if there’s ever any concern. Note that running in low-power mode makes this even less of an issue.

In a production implementation, it might be prudent to switch the 5V rail to a switching power supply as long as the voltage supplied to the A/D converter is carefully designed to be ripple free (low-pass filters likely required). Admitidly, the design shown is over cautious about noise so there is opportunity for circuit reduction if this design is ever to be taken into actual production. Addtionally, as mentioned in section 9.3.2 “HARDWARE”, a production implementation would likely eliminate the external power supply and run off the Accompli 009 battery directly which would be another motivation to re-visit the power distribution design.

APPENDIX B: SOFTWARE LISTING

```
/**************************************************************************
 *
 *        Thesis Research Project
 *             Biosensors
 *          Dr. William Hunt
 *            Fall, 2003
 *
 *    Project: build a working Spreeta BioSensor
 *                w/wireless notification
 *
 *************************************************************
 *
 * Module Name:
 *      spreeta.c
 *
 *  Creator:
 *      Author: dsommers
 *      Date: 2/20/2001
 *
 *  Purpose:
 *      Continuously transfer the TI Spreeta device to the LCD
 *
 *  Note: the algorithm attempts to do most of the overhead processing when
 *        the pixel value equals 0 or when the user changes modes because
during
 *        those moments, we're not in the middle of gathering data from the
 *        Spreeta and thus the timing is not nearly so critical...
 *
 * Revision A: Add capability to turn off Spreeta's LED as we seem to be
boiling
 *             away the liquid on the sensor.  This will save power as well as
 *             reduce the energy absorbed into the gold surface
 *
 * Revision B: add user inputs for controlling low power vs. high speed
 *             and adjustable integration period.
 *             June 7, 2001 DRS
 *
 * Revision C: show the change in the waveforms as red lines while keeping
 *             the black lines so it's easier to see change.
 *             July 12, 2001 DRS
 *
 * Revision D: smooth the waveforms so they don't jump around so much
 *             while the sensor is running.  Also useful to eliminate "rogue"
 *             detector cells
 *             October 10, 2001 DRS
 *
 * Revision E: add the threshold cursors to be able to detect when the
 *             sensor has changed significantly
 *             February 15, 2002 DRS
 *
 * Revision F: send wireless notification upon violation of threshold
 *             May 15, 2002 DRS
 *
 ***********************************************************************/


#define  VFMAPI_H
#define  GSM
#include <features.h>
#include <stdio.h>
#include <stdlib.h>
#include <types.h>

#include "mbatypes.h"
#include "mbadefs.h"
#include "mbaproto.h"
```

```
#include <modes.h>
#include <signal.h>
#include <string.h>
#include <airrun.h>
#include <thread.h>
#undef PWM_PERIOD

#include <sgstat.h>
#include <cglob.h>
#include <process.h>
#include <as_sigs.h>
#include <error.h>

#include <common.h>
#include <event.h>
#include <os_proto.h>
#include <fis.h>
#include <memory.h>
#include <module.h>
#include <emerald.h>
#include <regs_vz.h>
#include <compress.h>
#include <math.h>
#undef SIG_REORG
#include <mdsdefs.h>


#undef ERROR

/* function prototypes */
void CopyBitmapToScreen(char *FileBitmapName);
void display_absolute_screen(void);
void PrintTextToScreen( int16 y, int16 x, string txt );
void tmode_nopause(void);
void SampleSpreeta(int CaptureMode);
int ProcessUserInput( u_char input_data, int *algorithm_mode );
fisID_t BuildSMSMessage(char *AddressList, char *MessageText);
void SpawnNotification(void);
int CapturePopUpText(char **CaptureString, Point_t CurrPoint);

#define TOTAL_SPREETA_PIXELS 128
#define SPREETA_RESET_CLOCKS 18
#define RIU_MAX 1.368
#define RIU_MIN     1.320
#define WELCOME_SCREEN_FILE                   "MODULE:welcome.rob"
#define SIMULATION_ABSOLUTE_FILE  "MODULE:sim_abs.rob"
#define SIMULATION_DIFFERENTIAL_FILE "MODULE:sim_diff.rob"
#define SLEEP_1_SECOND 100
#define SLEEP_HALF_SECOND 50

#define DRAW_ABSOLUTE     1000
#define DRAW_ZOOM         1001
#define DRAW_DIFFERENTIAL 1002

#define CONTINUE_ALGORITHM 2000
#define RESTART_ALGORITHM  2001
#define QUIT_ALGORITHM       2002

/* define the return values for the cursor interference checks */
#define NO_INTERFERENCE 3000
#define ERASURE_COMPLETE 3001
/* define the modes for spreeta capture */
#define CAPTURE_PIXELS        3001
#define BEGIN_INTEGRATION 3002

#define TEXT_HEIGHT       8
#define TEXT_WIDTH        8
```

```
#define SPACE_BETWEEN_TEXT_LINES 1
#define TEXT_LINE_SPACING (TEXT_HEIGHT + SPACE_BETWEEN_TEXT_LINES)
#define CURSOR_WIDTH      TEXT_WIDTH - 4
#define CURSOR_HEIGHT     TEXT_HEIGHT - 2

/* all of the locations for the draws */
#define Y_TITLE 10
#define X_TITLE 70
#define X_VERT_AXIS  45
#define X_VERT_LABEL 10
#define X_TIME_LABEL (X_VERT_AXIS + 19)
#define Y_TOP_VERT_LABEL 25
#define VERTICAL_AXIS_TEXT "Intensity"
#define Y_VERT_LABEL_SPACING  30
#define Y_2ND_VERT_LABEL (Y_TOP_VERT_LABEL + Y_VERT_LABEL_SPACING)
#define Y_3RD_VERT_LABEL (Y_2ND_VERT_LABEL + Y_VERT_LABEL_SPACING)
#define Y_BOT_VERT_LABEL (Y_3RD_VERT_LABEL + Y_VERT_LABEL_SPACING)
#define Y_HORZ_AXIS_VALUES (Y_BOT_VERT_LABEL + TEXT_HEIGHT + 4)
#define Y_HORZ_AXIS_LABEL (Y_HORZ_AXIS_VALUES + TEXT_LINE_SPACING+1)
#define X_HORZ_AXIS_LABEL (X_VERT_AXIS+28)
#define X_SAMPLE_LABEL (X_HORZ_AXIS_LABEL + 50)
#define X_SAMPLE_COUNT 76
#define Y_SAMPLE_COUNT_LABEL (Y_HORZ_AXIS_LABEL + TEXT_LINE_SPACING+2)
#define STATUS_BAR_HEIGHT TEXT_HEIGHT
#define Y_HORZ_AXIS (Y_BOT_VERT_LABEL + TEXT_HEIGHT)
#define HORIZONTAL_AXIS_TEXT "Incident Angle"

#define X_AVG_MIN_MAX_LABEL 180
#define Y_AVG_LABEL (Y_TOP_VERT_LABEL + 7)
#define Y_MAX_LABEL (Y_AVG_LABEL + TEXT_LINE_SPACING + 15)
#define Y_MIN_LABEL (Y_MAX_LABEL + TEXT_LINE_SPACING*2 + 15)
#define X_AVG_MIN_MAX_VALUE X_AVG_MIN_MAX_LABEL

#define TOP_GRAPH_BOUNDARY     Y_TOP_VERT_LABEL+TEXT_HEIGHT
#define BOTTOM_GRAPH_BOUNDARY  Y_BOT_VERT_LABEL+TEXT_HEIGHT
#define POPUP_TOP_X 15
#define POPUP_TOP_Y 20
#define POPUP_BOTTOM_X 224
#define POPUP_BOTTOM_Y 145
#define POPUP_BOARDER_THICKNESS 5
#define ERROR_NO_TEXT_CAPTURE 911
#define MAX_POPUP_TEXT_LINES 20
#define MAX_POPUP_LINE_LENGTH 200

/* define some of the purrrrdy collers */
#define COLOR_BLACK       3
#define COLOR_WHITE       0
#define COLOR_TRANSPARENT 4
#define COLOR_BLUE        8
#define COLOR_BLUE_GREEN  31
#define COLOR_RED         171
#define COLOR_YELLOW      249
#define COLOR_GREEN       33
#define COLOR_DK_GREEN    111
#define COLOR_BROWN       193
#define COLOR_PRUPLE      46
#define COLOR_BURGUNDY    87
#define COLOR_LIGHT_RED   220

/* leave some room above and below the zoom inital values */
#define SCREEN_MAX_OFFSET 40
#define SCREEN_MIN_OFFSET 40

/* keep the TICKS PER PIXEL an even power of two to avoid floating point */
#define DIFFERENTIAL_ADC_TICKS_PER_SCREEN_PIXEL 8

/* A/D converter value corresponding to Spreeta's maximum output */
```

```
#define SPREETA_ADC_3V_VALUE 0xFFF
#define SPREETA_ADC_2V_VALUE ((SPREETA_ADC_3V_VALUE * 2) / 3)

/* define the different handles for styles of text */
#define NORMAL                    0
#define EMPHASIZED                1
#define LARGE                     2
#define LGEMPHASIZED              3
#define CAPTION                   5
#define HILITCAPTION              6
#define CONTROL                   7
#define HILITCONTROL              8
#define INACTCONTROL              9
#define LGCONTROL                10
#define LGHILITCONTROL           11
#define LGINACTCONTROL           12
#define STATUSBAR                13
#define SYSTEMCONSOLE            14
#define MINIME_TEXT              15
#define CLOCK_TEXT              16
#define NELSON_PROP_TEXT         17
#define GARNER_SAN_SERIF_TEXT    18
#define MASON_SAN_SERIF_TEXT     19
#define SPITFIRE_TEXT           20
#define SPITFIRE_LARGE_NONPROP   21

/* SPI register control data */
/* note: the book is wrong! the transmit & receive registers are 16 bits wide
*/
#define SPI_RECEIVE  ((volatile u_int16*)0xFFFFF700)
#define SPI_TRANSMIT ((volatile u_int16*)0xFFFFF702)
#define SPI_CONTROL  ((volatile u_int16*)0xFFFFF704)
#define SPI_STATUS   ((volatile u_int16*)0xFFFFF706)
#define SPI_SAMPLE   ((volatile u_int16*)0xFFFFF70A)
#define SPI_EXCHANGE *SPI_CONTROL |= 0x0100

/* configure  SPREETA: 3 bit data transfer; 8MHz clock rate (sysclk/4) */
#define SPI_SET_SPREETA   *SPI_CONTROL &= ~(0xE00F); *SPI_CONTROL |= 0x0003
/* configure MAXIIM: 13 bit data transfer; 2MHz clock rate (sysclk/16) */
#define SPI_SET_MAXIM     *SPI_CONTROL &= ~(0xE00F); *SPI_CONTROL |= 0x400C
#define WAIT_EXCHANGE while ((*SPI_STATUS & 0x0008) == 0) {;}

/* define the SPI data for the shift register */
#define MAXIM_CS          0x0000
#define SPREETA_START     0x0006
#define SPREETA_CLOCK     0x0005
#define SPREETA_BOTH      0x0007
#define SHIFT_REG_ALL_OFF 0x0004
#define SPREETA_LED_ON    0x0008

/* interrupt controller definitions */
#define INTERRUPT_MASK_REGISTER  ((volatile u_int32*)0xFFFFF304)
#define MASK_ALL_INTERRUPTS 0xFFFFFFFF
#define MASK_INTERRUPTS      gSavedInterruptMaskValue =
*INTERRUPT_MASK_REGISTER; \
                             *INTERRUPT_MASK_REGISTER = MASK_ALL_INTERRUPTS
#define RESTORE_INTERRUPTS   *INTERRUPT_MASK_REGISTER =
gSavedInterruptMaskValue

/* PWM definitions */
#define PWM1_PERIOD  ((volatile u_int8*)0xFFFFF504)
#define PWM1_COUNTER ((volatile u_int8*)0xFFFFF505)
#define PWM1_SAMPLE  ((volatile u_int8*)0xFFFFF503)
#define PWM1_CONTROL ((volatile u_int16*)0xFFFFF500)
/* for 32mA LED current, ideal integration period is ~25mSec
   1/32.768KHz = 30.52uSec
   25mSec = 30.52uSec * 820
   Use default count set to 128 to insure room for increase/decrease
   25mSec/128 = 195.3uSec
   Thus divide the 32KHz clock by: 195.3/30.52 = 6.4 = ~6
*/

/*  PWM1_CONTROL = 0x8310;           1 32KHz Crystal clock source
                              0000011 prescaler of 3 for 195uSec tick rate
                                    0 Interrupt Request
                                    0 Disable Interrupt
                                    0 FIFO Available
                                   1 Enabled
                                  00 No samples repeated
                                  00 Divide Clock Source by 2 for 61.1uSec
Tick Rate */

/* note: must re-init PWM1 every usage because keyboard beeps & clicks will
change it */
/*
#define START_COUNTER *PWM1_CONTROL = 0x8310; *PWM1_PERIOD = 0xFF;
*PWM1_SAMPLE = 0xFF
#define RESET_COUNTER (*PWM1_CONTROL &= ~(0x0010))
*/
/*  32.768KHz/6 * 25mSec = 136 sample count:
    each increment/decrement of the integration timeout
    increases/decreases the integration period by 183.1uSec
*/
/*
#define DEFAULT_INTEGRATION_TIMEOUT (u_int8)136
*/

/* Note: can't get enough resolution out of the PWM, so use Timer2 set to
10uSec */
#define TMR2_COUNTER  ((volatile u_int16*)0xFFFFF618)
#define TMR2_PRESCALE ((volatile u_int16*)0xFFFFF612)
#define TMR2_CONTROL  ((volatile u_int16*)0xFFFFF610)

/* want 1uSec resolution: use SYSCLK/16 = 32MHz/16 = 2MHz
   1uSec => 1MHz
   2MHz/1MHz = 2 = prescale 2
   for 10mSec delay 10mSec/1uSec = 10000 = Timeout Value
   Integration increment = 50uSec => 50
*/

/*  TMR2_CONTROL = 0x0105;      1 Free Running mode
                               00 Disable Capture
                                0 Active-low Pulse
                                0 Disable Compare Interrupt
                              010 SYSCLK/16
                                1 Timer Enabled
*/


#define START_COUNTER *TMR2_CONTROL = 0x0105; *TMR2_PRESCALE = 2;
#define RESET_COUNTER (*TMR2_CONTROL &= ~(0x0001))

#define DEFAULT_INTEGRATION_TIMEOUT (u_int16)10000
#define TIME_PERIOD_OF_5mSEC 5000
#define INTEGRATION_INCREMENT 50
/* Port J Definitions; used to control SPI_SS programmatically instead of SPI
driven */
#define PJ_DIRECTION ((volatile u_int8*)0xFFFFF438)
#define PJ_DATA      ((volatile u_int8*)0xFFFFF439)
#define PJ_PUEN      ((volatile u_int8*)0xFFFFF43A)
#define PJ_SELECT   ((volatile u_int8*)0xFFFFF43B)
#define SPI_CS_LOW   *PJ_DATA &= ~(0x08)
#define SPI_CS_HIGH  *PJ_DATA |=  (0x08)
```

```
/* Port B Definitions; used to control Alert Volume (e.g. turn it off so we
can use TIMER2 for timing) */
#define PB_DIRECTION ((volatile u_int8*)0xFFFFF408)
#define PB_DATA      ((volatile u_int8*)0xFFFFF409)
#define PB_PUEN      ((volatile u_int8*)0xFFFFF40A)
#define PB_SELECT    ((volatile u_int8*)0xFFFFF40B)
#define ALERT_VOLUME_OFF   *PB_DATA |= (0x80)
#define ALERT_VOLUME_MAX  *PB_DATA &=  ~(0x80)

/* FIS Type & ID */
#define BITMAP_FIS_TYPE 6
#define BITMAP_FIS_ID   18901
#define MAILOBJ_FIS_ID  18901

/* Speed Modes */
#define LOW_POWER_MODE  0x00000000
#define HIGH_SPEED_MODE 0xFFFFFFFF

/* Threshold Cursor Definitions */
#define CURSOR_SOLID  1
#define CURSOR_DASHED 2
#define CURSOR_DEFAULT_COLOR  COLOR_GREEN;
#define CURSOR_SELECTED_COLOR COLOR_BLUE_GREEN;
#define MAX_CONSECUTIVE_THRESHOLD_VIOLATIONS 3

struct HorizontalCursor{
	int PositionDifferential;  /* pixel position of the cursor on the
Differential Screen */
	int PositionAbsolute;        /* pixel position of the cursor on the
Absolute Screen */
	int Color;      /* line color */
	int Width;      /* line width in pixels */
	int Style;   /* line style: e.g. dashed, solid, etc.) */
	struct HorizontalCursor *Next;   /* next cursor in the linked list */
};

/* notificaiton status states */
#define NOTIFICATION_DORMANT         0
#define NOTIFICATION_ENABLED         1
#define NOTIFICATION_IN_PROGRESS  2

struct SMSNotification{
	fisID_t SMS_ID;   /* the FIS ID of the SMS struct */
	int Sent;
	char *Address;  /* the e-mail address for this notification */
	struct SMSNotification *Next;   /* next SMS notification in the linked
list */
};

/* Global variables */
u_int16  gCurrentSpreetaValue[TOTAL_SPREETA_PIXELS];
u_int16  gBaselinedSpreetaValue[TOTAL_SPREETA_PIXELS];  /* used for
differential mode */
u_int16  gPreviousScreenValue[TOTAL_SPREETA_PIXELS];   /* used to erase
previous screen pixel */
u_int16  gCapturedAbosluteScreenValue[TOTAL_SPREETA_PIXELS];  /* used to show
change since last capture */
u_int16  gCapturedZoomScreenValue[TOTAL_SPREETA_PIXELS];     /* used to show
change since last capture */
int      gSpreetaAvgValue;  /* average of all Spreeta Values */
int      gSpreetaMaxValue;
int      gSpreetaMaxValueLocation;
int      gSpreetaMinValue;
int      gSpreetaMinValueLocation;
int       gScreenMaxValue; /* used for calculating the differential and zoom
maximum and minimum */
int      gScreenMinValue;
int      gSampleCount;  /* tracks the count for statistical fun... */
float    gConvertADCToScreen;  /* used to eliminate a FloatingPoint divide
during the main loop */
path_id  keypad_path;       /* path number to /kpad */
u_int32  gSavedInterruptMaskValue; /* data read from the interrupt mask
register */
int       gSpreetaPixelsBaselined;  /* used to ensure we don't try to run
differential until we have a capture */
int      gDifferentialTicksPerPixelPower; /* keep an even power of 2 */
int       gIntegrationPeriod; /* the time the LED is on and we're integrating
*/
u_int32  gHighSpeedMode; /* enables/disables the sleep period for low power
vs. high speed */

/* declare 2 horizontal cursors in a circular linked list */
struct HorizontalCursor gHorizontalCursor1, gHorizontalCursor2;
struct HorizontalCursor *gHorizontalCursorActive;  /* pointer to the active
cursor */
struct HorizontalCursor gDifferentialCenterLine;  /* red line is implemented
as a cursor */

/* globals for tracking notificaiton */
int   gNotificationStatus;        /* track if there's an SMS to send out upon
detection of threshold violaiton */
int   gThresholdCrossedCount;     /* count of number of consecutive
violations of the threshold (e.g. violation filter) */
struct SMSNotification *gpSMSNotificationList; /* pointer to linked list of
SMS messages */

extern RenderingContext_t DefaultRenderingContext;  /* defined and manitained
by the LCD driver */

/* #define NOISY */ /* Uncomment if you want printf's out the A009 datacable
*/
/* #define VERY_NOISY */ /* Uncomment if you want a whole lot more printf's */
/*

*******************************************************************************
*****
 * Function Name: doPerformReorg
 *
 * Inputs:
 *         count     how many reorgs are done
 *         reorgType whether to reorg the NRB area or not
 * Outputs:
 *         none
 * Return:
 *              none
 * Description: This function is just a stub function to make Spreeta compile
correctly
 * The actual code to the actual function is in vm/thread/event.c.
* Cautions:
 *

*******************************************************************************
*****
 */
void doPerformReorg( u_int16 count, u_int16 reorgType )
{
}

/******************************************************************************
****
 *
 * PrintTextToScreen()
 *
 * takes the text in the string txt and draws it to the screen at the
 * (x,y) location using porportional font
 *

*****************************************************************************
***/
void PrintTextToScreen( int16 y, int16 x, string txt )
{
    Point_t     currPoint;
      WinBase_t   TextWindow;

    if ((y==-1) && (x==-1))
    {
          ClearScreen(&DefaultRenderingContext,COLOR_WHITE);
          x = y = 0;
    }

    currPoint.yCoord = y;
    currPoint.xCoord = x;
      TextWindow.Rect.RX1 = 0;   /* define the region of the text area as the
full screen */
      TextWindow.Rect.RY1 = 0;
      TextWindow.Rect.RX2 = 239;
      TextWindow.Rect.RY2 = 159;

    WinDrawTextT(&DefaultRenderingContext, &TextWindow, &currPoint,
COLOR_BLACK, COLOR_WHITE, COLOR_TRANSPARENT,
      MINIME_TEXT, txt, strlen((const char *)txt));
    SetTextCursor(&currPoint);   /* update the cursor point to the end of the
text line */

    /* This only does non-proportional fonts
    SetTextCursor(&currPoint);
    DrawText( txt, strlen((const char *)txt) );     */

}

/****************************************************************************
****
 *
 * UpdateValueOnScreen()
 *
 * Erases the text field on the screen so we can update in place.  Reason
 * for this is the proportional Font leaves gaps between the characters and
 * if un-erased, the old data will show through the cracks between the
 * characters and the field will become a real mess
 *
*****************************************************************************
***/
void UpdateValueOnScreen( int16 y, int16 x, string txt )
{
    Point_t     CurrPoint;

      CurrPoint.xCoord = x;
      CurrPoint.yCoord = y;
      SetDrawingCursor(&CurrPoint);
      CurrPoint.xCoord += ((TEXT_WIDTH-2)*strlen((char *)txt))+3;
      CurrPoint.yCoord += TEXT_HEIGHT;
      Box(&DefaultRenderingContext, &CurrPoint, 1, COLOR_WHITE);

      PrintTextToScreen(y, x, txt);

}

/****************************************************************************
****
 *
 * PrintTextCursor()
 *
 * print the cursor at the end of the text string
 *

******************************************************************************
***/
void PrintTextCursor( void )
{
    Point_t     CurrPoint;

       GetTextCursor(&CurrPoint);
       SetDrawingCursor(&CurrPoint);
       CurrPoint.xCoord += CURSOR_WIDTH;
       CurrPoint.yCoord += CURSOR_HEIGHT;
       Box(&DefaultRenderingContext, &CurrPoint, 1, COLOR_BLACK);

}

/*****************************************************************************
****
 *
 * EraseTextCursor()
 *
 * Erases the text cursor so we can print the next character
 *

******************************************************************************
***/
void EraseTextCursor( void )
{
    Point_t     CurrPoint;

       GetTextCursor(&CurrPoint);
       SetDrawingCursor(&CurrPoint);
       CurrPoint.xCoord += CURSOR_WIDTH; /* erase the cursor */
       CurrPoint.yCoord += CURSOR_HEIGHT;
       Box(&DefaultRenderingContext, &CurrPoint, 1, COLOR_WHITE);

}
/*****************************************************************************
****
*
 * EraseLastCharacter()
 *
 * Erases the last character in the line of text and the cursor
 *

******************************************************************************
***/
void EraseLastCharacter( Point_t CurrPoint, string txt )
{

       Point_t ErasePoint;

       EraseTextCursor();
       /* back up at least one character space */
       GetTextCursor(&ErasePoint);
       ErasePoint.xCoord -= TEXT_WIDTH;
       SetDrawingCursor(&ErasePoint);
       ErasePoint.xCoord += TEXT_WIDTH;
       ErasePoint.yCoord += TEXT_HEIGHT;
       Box(&DefaultRenderingContext, &ErasePoint, 1, COLOR_WHITE);
       txt[strlen((const char *)txt)-1] = 0x00; /* erase the last character
out of the buffer */
       PrintTextToScreen(CurrPoint.yCoord, CurrPoint.xCoord, txt);
       PrintTextCursor();
}

/******************************************************************************
****
 *
 * PrintNextCharacter()
 *
 * Prints the next character
 *
 * 1) erase the cursor
 * 2) print the new character by printing the entire string
 * 3) draw the cursor at the new position
 *

*******************************************************************************
***/
void PrintNextCharacter( Point_t CurrPoint, string txt )
{

      #ifdef NOISY
    printf("Entering PrintNextCharacter with string \"%s\"\n",txt);
      #endif

      EraseTextCursor();
      PrintTextToScreen(CurrPoint.yCoord, CurrPoint.xCoord, txt);
      PrintTextCursor();

}

 /*

*******************************************************************************
*****
 * Function Name: getTicks
 *
 * Simply returns the number of ticks in the OS.  A tick is 10mSec
 *

*******************************************************************************
*****
 */
u_int32 getTicks( void )
{
      u_int32 time, date, ticks;
      u_int16 day;

      _os9_getime( 3,                 /* julian with ticks */
                   &time,
                   &date,
                   &day,
                   &ticks );
      return ( (time * (ticks >> 16)) + (ticks & 0xFFFF ));

}

/*

*******************************************************************************
*****
 * Function Name: UpdateDateOnScreen()
 *
 * Inputs:
 *       *piHour- 0-23, where 0 indicates midnight and 12 is noon
 *       *piMinute- 0-59
 *       *piSecond- 0-59
 *
 * Outputs:
```

```
 *        format- used in call to _os9_getime
 *            0 - Gregorian format
 *            1 - Julian format
 *            2 - Gregorian format w/ticks
 *            3 - Julian format w/ticks
 *        *piHour- 0-23, where 0 indicates midnight and 12 is noon
 *        *piMinute- 0-59
 *        *piSecond- 0-59
 *
 * Return:
 *     OK- the time was successfully received
 *     EVTERR_TIMER_READ_FAILURE- an error occurred when _os9_getime was
called
 *
 * Description: Retrieves the Time Of Day from the internal real-time clock.
It is
 *   intended to be called by FLEX Script programs.
 *
 * Cautions: Use the calls to _os9 because they work better than the _os calls
 *

******************************************************************************
*****
 */
ErrorCode UpdateDateOnScreen ( void )
{
    u_int32 format = GREGORIAN_FORMAT;
    u_int32 time;
    u_int32 date;
    u_int16 day;   /* we get the day & throw it away */
    u_int32 ticks;  /* we get the ticks & throw it away */
      u_int16 piYear, piMonth, piDate;
      char    pScreenValue[50];
    ErrorCode return_val = OK;

    if ((_os9_getime ( format, &time, &date, &day, &ticks )) != OK)
    {
        return_val = EVTERR_TIMER_READ_FAILURE;
    }
    else
    {
        piYear = (u_int16)(date >> 16);   /* pull year, month & day out of 32-
bit integer */
        date &= 0x0000FFFF;
        piMonth = (u_int16)(date >> 8);
        date &= 0x000000FF;
        piDate = (u_int16)date;

             sprintf(pScreenValue,"%d/%d/%02d",piMonth,piDate,piYear-2000);

      UpdateValueOnScreen(Y_SAMPLE_COUNT_LABEL,X_VERT_LABEL,(string)pScreenVa
lue);
    }


      return(return_val);
}

/*

******************************************************************************
*****
 * Function Name: UpdateTimeOnScreen()
 *
 * Inputs:
 *        *piHour- 0-23, where 0 indicates midnight and 12 is noon
 *        *piMinute- 0-59
 *        *piSecond- 0-59
 *
 * Outputs:
 *        format- used in call to _os9_getime
 *            0 - Gregorian format
 *            1 - Julian format
 *            2 - Gregorian format w/ticks
 *            3 - Julian format w/ticks
 *        *piHour- 0-23, where 0 indicates midnight and 12 is noon
 *        *piMinute- 0-59
 *        *piSecond- 0-59
 *
 * Return:
 *     OK- the time was successfully received
 *     EVTERR_TIMER_READ_FAILURE- an error occurred when _os9_getime was
called
 *
 * Description: Retrieves the Time Of Day from the internal real-time clock.
It is
 *   intended to be called by FLEX Script programs.
 *
 * Cautions: Use the calls to _os9 because they work better than the _os calls
 *

******************************************************************************
*****
 */
ErrorCode UpdateTimeOnScreen ( void )
{
    u_int32 format = GREGORIAN_FORMAT;
    u_int32 time;
    u_int32 date;
    u_int16 day;   /* we get the day & throw it away */
    u_int32 ticks;  /* we get the ticks & throw it away */
      u_int16 piHour, piMinute, piSecond;
      char    pScreenValue[50];
    ErrorCode return_val = OK;

    if ((_os9_getime ( format, &time, &date, &day, &ticks )) != OK)
    {
        return_val = EVTERR_TIMER_READ_FAILURE;
    }
    else
    {
        piHour = (u_int16)(time >> 16);  /* pull Hour, Minutes & Seconds out
of 32-bit integer */
             piHour &= 0x001F; /* bug: lop off the top bits because sometimes
hours >32 */
        time &= 0x0000FFFF;
        piMinute = (u_int16)(time >> 8);
        piSecond = (u_int16)(time & 0x0000FF);

             sprintf(pScreenValue,"%d:%02d:%02d",piHour,piMinute,piSecond);

      UpdateValueOnScreen(Y_SAMPLE_COUNT_LABEL,X_TIME_LABEL,(string)pScreenVa
lue);
    }


      return(return_val);
}

/***************************************************************
 *
 * DeleteAllSpreetaMailObjs()
 *
 * search through FIS for MailObjs that were created by this Spreeta
 * code and delete them.  They would be defined as orphans otherwise...
 *
 * Returns:
 ***************************************************************/
void DeleteAllSpreetaMailObjs(void)
{
	fisID_t fisID;

	#ifdef NOISY
	printf("entering DeleteAllSpreetaMailObjs\n");
	#endif

	/* create mailobjects in a seperate place to keep track of them and
distroy them */
	fisID = MAILOBJ_FIS_ID;

	/* search for all Spreeta-Created MailObjs and delete them... */
	while (FISDelete(INCOMING_MAIL_TYPE, fisID) != E_ITEM_NOT_FOUND)
	{
		#ifdef NOISY
		printf("Deleting MailObj ID: %d\n",fisID);
		#endif
		fisID++;
	}
}

/************************************************************
 *
 * InitMessagingSubSYstem()
 *
 * this routine set up the WhiteCap for sending messages
 *
 * Returns:
 *    SUCCESS - everything fine
 *    FAILURE - something went wrong
 ***************************************************************/
int InitMessagingSubSystem(void)
{
	eventmsg InitWCEventMsg;   /* used to enable phone mode */
	process_id SpreetaPID;
	process_id MDSPID;

	DeleteAllSpreetaMailObjs();  /* clean out all lingering orphans */
	name2pid("spreeta", &SpreetaPID);
	name2pid("mmds", &MDSPID);

	InitWCEventMsg.FromPid = SpreetaPID;
	InitWCEventMsg.EventCode = EVENT_BATTERYDOOR_OPEN;
	InitWCEventMsg.asTask = 0;
	InitWCEventMsg.Param1 = SUBEV_PS_ME_ERROR_REPORT;
	InitWCEventMsg.Param2 = 0; /* not used */
	#ifdef NOISY
	printf("Sending SUBEV_PS_ME_ERROR_REPORT (0x%x) event to
MDS\n",SUBEV_PS_ME_ERROR_REPORT);
	#endif
	SendMsg(MDSPID, &InitWCEventMsg);

	InitWCEventMsg.Param1 = SUBEV_MS_MEMORY_STATUS_AVAILABLE;
	#ifdef NOISY
	printf("Sending SUBEV_MS_MEMORY_STATUS_AVAILABLE (0x%x) event to
MDS\n",SUBEV_MS_MEMORY_STATUS_AVAILABLE);
	#endif
	SendMsg(MDSPID, &InitWCEventMsg);

	InitWCEventMsg.EventCode = EVENT_TRANSMITTER_CNTRL;
	InitWCEventMsg.Param1 = POWER_WHITECAP_PHONE_MODE;
	#ifdef NOISY
```

```
       printf("Sending EVENT_TRANSMITTER_CNTRL (0x%08x) event to
MDS\n",EVENT_TRANSMITTER_CNTRL);
       #endif
       SendMsg(MDSPID, &InitWCEventMsg);

    while (GetMsg(&InitWCEventMsg) == SUCCESS)
       {
              #ifdef NOISY
              printf("Received event 0x%x; Param1: 0x%x from
MDS\n",InitWCEventMsg.EventCode, InitWCEventMsg.Param1);
              #endif
       }
       return(SUCCESS);
}


/**************************************************************
 *
 * initalize_system()
 *
 * this routine will:
 *
 * 1) initalize the LCD (bitmaps & text)
 * 2) initalize the keypad input
 * 3) initalize the Spreeta hardware
 **************************************************************/
int initalize_system(int *pAlgorithmMode)
{
    path_id        keypad_path;       /* path number to /kpad */
    kpad_mode_t    my_mode;
    kpad_msgmode_t my_msgmode;
       int            algorithm_status;
       volatile int   i;                  /* loop counter and general variable;
poorly named */
       ErrorCode      error;              /* used for debug */

       #ifdef DEBUG_MESSAGE_SEND
       fisID_t fisID;

       fisID = BuildSMSMessage("dan.sommers@motorola.com", "Danzadope");
       SpawnNotification(fisID);
       #endif

       #ifdef NOISY
    printf("Entering Device and Spreeta initalization routine.\n");
       #endif

    tmode_nopause();     /* turn off screen pause mode for debug output to
stdout */
    lcd_suspend_time(0); /* disable screen timeout */
    suspend_time(0);       /* disable system sleep */

       InitializeFileSystem();    /* allows access to files; most FW calls
rely on this */
       /* InitializeDBMS(); */           /* allows access to main database */
    InitializeDisplay(); /* initalize the LCD hardware */
       Init_NonVolatile_Date();

       #ifdef VERY_NOISY
    printf("DefaultRenderingContext.Device.DisplayHeight = %d;
DefaultRenderingContext.Device.DisplayWidth = %d\n",

DefaultRenderingContext.Device.DisplayHeight,DefaultRenderingContext.Device.Di
splayWidth);
    printf("DefaultRenderingContext.Device.BitsPerPixel = %d;
DefaultRenderingContext.Device.LineWidth = %d\n",
DefaultRenderingContext.Device.BitsPerPixel,DefaultRenderingContext.Device.Lin
eWidth);
    printf("DefaultRenderingContext.Device.PixelPerByte =
%d\n",DefaultRenderingContext.Device.PixelPerByte);
    printf("DefaultRenderingContext.Device.DisplayMemory =
0x%08x\n",DefaultRenderingContext.Device.DisplayMemory);
       #endif

    CopyBitmapToScreen(WELCOME_SCREEN_FILE);
    lcd_baklit_on(); /* turn on the backlight for maximum User Experiance
pleasure */

       /* initalize the text system */
    error = InitText();
    SelectFont(MINIME_TEXT);
    SetTextColor(COLOR_BLACK, COLOR_WHITE);  /* Black text on White Background
*/

    /*****************************************
     *   initialize Kepyad Driver
     *****************************************/
    if ( (error = _os_open ("/kpad", FAM_READ+FAM_WRITE, &keypad_path)) !=
SUCCESS)
    {
          #ifdef NOISY
        printf("Can't open path to /kpad (error 0x%x)\n", error);
        #endif
    }
    else
      {
          #ifdef NOISY
        printf("Opened path to /kpad (path # = %d)\n", keypad_path);
          #endif
      }

    /*****************************************
     *   Set Processing Mode for Keypad:
        *
        *   COOKED => we want only keys that have
        *             been processed as a single
        *             key value return
     *****************************************/
    if ((error = kpad_set_mode(keypad_path, COOKED)) != SUCCESS)
    {
        printf("kpad_set_mode() error (0x%2x)\n", error);
        exit(0);
    }
    if ((error = kpad_get_mode(keypad_path, &my_mode)) != SUCCESS)
    {
        printf("kpad_get_mode() error (0x%2x)\n", error);
        exit(0);
    }
    if (my_mode != COOKED)
    {
        printf("kpad_get_mode() error (returned wrong mode)\n");
        exit(0);
    }

    /*******************************************************
     *   Set Keypad Signalling Mode
        *
        *   IPMQ => all keys will queue up into the input Queue
        *   and wait patiently until we get around to processing them
     *******************************************************/
    if ((error = kpad_set_msgmode(keypad_path, IPMQ)) != SUCCESS)
    {
        printf("kpad_set_mode() error (0x%2x)\n", error);
        exit(0);
    }
    if ((error = kpad_get_msgmode(keypad_path, &my_msgmode)) != SUCCESS)
    {
        printf("kpad_get_mode() error (0x%2x)\n", error);
        exit(0);
    }
    if (my_msgmode != IPMQ)
    {
        printf("kpad_get_mode() error (returned wrong mode)\n");
        exit(0);
    }

      kpad_set_click (keypad_path, CLICK_OFF); /* must turn off clicks
because we re-use PWM1
                                           for the timing loop */
    /*
     * Initialize InterProcess Messaging Queue
     */
    if ((error = InitMsg(10, FALSE)) != SUCCESS)
    {
        printf("InitMsg Failed rc = %x\n", error);
        exit(0);
    }

      /* initalize the arrays for first-time through calculations */
      for (i=0; i<TOTAL_SPREETA_PIXELS; i++)
      {
            gCurrentSpreetaValue[i] = SPREETA_ADC_3V_VALUE/2;     /* average
value will be average volatge */
        gBaselinedSpreetaValue[i] = gCurrentSpreetaValue[i];
        gPreviousScreenValue[i] = Y_HORZ_AXIS - (int)(gCurrentSpreetaValue[i]
* (float)3/SPREETA_ADC_3V_VALUE);
            gCapturedAbosluteScreenValue[i] = 0; /* guaranteed to be off-
graph */
            gCapturedZoomScreenValue[i] = 0;  /* guaranteed to be off-graph
*/
      }
      gSpreetaPixelsBaselined = FALSE; /* no Differential Mode until capture
*/
      gSpreetaMaxValue = 0;
      gSpreetaMinValue = SPREETA_ADC_3V_VALUE;
      gDifferentialTicksPerPixelPower = 3; /* defaults to 2^3=8 Ticks per
Pixel */
      gIntegrationPeriod = DEFAULT_INTEGRATION_TIMEOUT;
      gHighSpeedMode = HIGH_SPEED_MODE; /* default to false so we can adjust
the system before sampling */

      /* initalize SPI driver */

      *PJ_SELECT &= ~(0x0F); /* enable MOSI-bit0; MISO-bit1; SPICLK1-bit2;
SPI_SS-bit3 as functions */
      *PJ_PUEN   &= ~(0x0F); /* turn off PULLUPS for MOSI; MISO; SPICLI1; SS
*/

      *SPI_CONTROL = 0x4642; /* 010 SysClock/16 => 2.06 MHz
                               00 DataReady Don't Care
                                1 SPI Master
                                1 SPI Interface Enabled
                                0 No Exchange
                                0 SS Polarity Active Low
                                1 SS Waveform Select Insert Pulse Between
Transfers
                                  0 Phase 0 Operation
                                  0 Active Hight Polarity
                                  0010 3-bit exchange (Spreeta Serial Shift
Register value */
       *SPI_SAMPLE = 0x0000;  /* 0 clocks between transfers */

       SPI_SET_SPREETA; /* initialize the Shift register outputs */
       *SPI_TRANSMIT = SHIFT_REG_ALL_OFF; /* load the FIFO to clear Spreeta's
Clock & Start; Set Maxim's CS High  */
       SPI_EXCHANGE;
       WAIT_EXCHANGE;
       i = *SPI_RECEIVE; /* read out the four bits of junk that came in */

       SampleSpreeta(BEGIN_INTEGRATION);      /* run through a capture to
initalize the system */

       /* initialize Volume */
       *PB_DIRECTION |= 0x80;  /* set ALERT Volume (PWMO) to an output */
       *PB_SELECT |= 0x80;     /* choose the I/O port function */
       ALERT_VOLUME_OFF;

       /* _os_sysdbg((void*)0xCAFEBABE, (void*)0xDEADBEEF);  */  /* use only
if rombug installed */

       /* initialize the horizontal cursors */
       gHorizontalCursor1.PositionDifferential = TOP_GRAPH_BOUNDARY +
(BOTTOM_GRAPH_BOUNDARY - TOP_GRAPH_BOUNDARY)/4;
       gHorizontalCursor2.PositionDifferential = TOP_GRAPH_BOUNDARY +
3*(BOTTOM_GRAPH_BOUNDARY - TOP_GRAPH_BOUNDARY)/4;

       gHorizontalCursor1.PositionAbsolute =
gHorizontalCursor1.PositionDifferential;
       gHorizontalCursor2.PositionAbsolute =
gHorizontalCursor2.PositionDifferential;

       /* set line with of the cursor to 2 */
       gHorizontalCursor1.Width = 2;
       gHorizontalCursor2.Width = 2;

       gHorizontalCursor1.Color = CURSOR_DEFAULT_COLOR;
       gHorizontalCursor2.Color = CURSOR_DEFAULT_COLOR;

       /* establish the linked list for the cursors */
       gHorizontalCursor1.Next = &gHorizontalCursor2;
       gHorizontalCursor2.Next = &gHorizontalCursor1;

       /* set the active cursor */
       gHorizontalCursorActive = &gHorizontalCursor1;
       gHorizontalCursorActive->Color = CURSOR_SELECTED_COLOR;

    /* turn off notification */
       gNotificationStatus = NOTIFICATION_DORMANT;
       gThresholdCrossedCount = 0;

    /* the center line for the differential screen is implemented as a cursor
*/
       gDifferentialCenterLine.PositionAbsolute = TOP_GRAPH_BOUNDARY +
(Y_BOT_VERT_LABEL - Y_TOP_VERT_LABEL)/2;
       gDifferentialCenterLine.PositionDifferential = TOP_GRAPH_BOUNDARY +
(Y_BOT_VERT_LABEL - Y_TOP_VERT_LABEL)/2;
       gDifferentialCenterLine.Color = COLOR_RED;
       gDifferentialCenterLine.Width = 2;
       gDifferentialCenterLine.Next = NULL;

       /* this takes a long time so let it get started... */
       InitMessagingSubSystem();

       /*
        * With welcome screen showing, wait for user to start program
```

```
         */
        #ifdef NOISY
        printf("So... press the \"any\" key and we'll get on with it...\n");
        #endif
        {
                eventmsg FirstMsg;
                int ContinueInput = TRUE;
                while ((ContinueInput == TRUE) && (ReadMsg(&FirstMsg) ==
SUCCESS)) /* blocking read */
                {
                        if (  (FirstMsg.EventCode == EVENT_KEY)
                           || (FirstMsg.EventCode == EVENT_SYSKEY)
                           || (FirstMsg.EventCode == EVENT_MODIFIER_STATE_CHANGE))
                        {
                                ContinueInput = FALSE;
                        }
                }
                #ifdef NOISY
                printf("Received Event %d (EVENT_KEY is
5)\n",FirstMsg.EventCode);
                #endif
        }
        algorithm_status = ProcessUserInput('s',pAlgorithmMode);
        display_absolute_screen();
        UpdateDateOnScreen();
        UpdateTimeOnScreen();
    return (algorithm_status);
}

/***************************************************************
 *
 * deinitalize_system()
 *
 * this routine will:
 *
 * 1) destroy the path to the keypad
 ********************************************************************/
 void deinitalize_system(void)
 {
    ErrorCode rc;

    _os_close (keypad_path);           /* _os_close() forces unlink of kpad */

       DeleteAllSpreetaMailObjs();  /* destroy all orphans lingering during
this session */
    rc = DeInitMsg();  /* close down the queue */
    if (rc != SUCCESS)
    {
        if (rc != ERROR_NOQ)  /*  ignore if InitMsg hasn't been called yet  */
        {
            printf("DeInitMsg failed: returned %x\n, rc");
        }
    }

       DeInitText();
    DeInitializeDisplay();

} /* end deinitalize_system() */

/***************************************************************
 *
 * CopyBitmapToScreen()
 *
 * draws some nice pictures and pride-generators to the screen
 *
 * The Welcome Screen is one big honker that covers the entire LCD
 * and has lots of good formatting...
 *
 *******************************************************************/
void CopyBitmapToScreen(char *BitmapFileName)
{
      Point_t   ScreenPosition;

      ScreenPosition.xCoord = 0; /* bitmaps drawn from (0,0) */
      ScreenPosition.yCoord = 0;

      #ifdef NOISY
    printf("Entering CopyBitmapToScreen to slap up: \"%s\"\n",BitmapFileName);
      #endif


FileBitmapToScreen(&DefaultRenderingContext,&ScreenPosition,(string)BitmapFile
Name);
}

/**************************************************************
 *
 * DrawHorizontalCursor()
 *
 * Simply draw a horizontal line across the screen
 *
 * Inputs:
 *     HC_ptr - pointer to the horizontal cursor structure
 *     DrawMode - differential vs. absolute
 *
 *
 *******************************************************************/
void DrawHorizontalCursor(struct HorizontalCursor *HC_ptr, int DrawMode)
{
      Point_t   CurrPoint;

      CurrPoint.xCoord = X_VERT_AXIS+1; /* don't write on top of the vertical
axis */
      switch (DrawMode)
      {
            case DRAW_DIFFERENTIAL:
                  CurrPoint.yCoord = HC_ptr->PositionDifferential;
                  break;
            case DRAW_ZOOM:
            case DRAW_ABSOLUTE:
                  CurrPoint.yCoord = HC_ptr->PositionAbsolute;
                  break;
            default: /* what the heck happened here?!? */
                  CurrPoint.yCoord = HC_ptr->PositionAbsolute;
                  break;
      }
      SetDrawingCursor(&CurrPoint);
      CurrPoint.xCoord += TOTAL_SPREETA_PIXELS-1;
      CurrPoint.yCoord += HC_ptr->Width - 1;
      Box(&DefaultRenderingContext, &CurrPoint, 1, HC_ptr->Color);

}


/**************************************************************
 *
 * EraseHorizontalCursor()
 *
 * Simply erase a horizontal line across the screen
 *
 * Inputs:
 *     HC_ptr - pointer to the horizontal cursor structure
 *     DrawMode - differential vs. absolute
 *
 *
 ******************************************************************/
void EraseHorizontalCursor(struct HorizontalCursor *HC_ptr, int DrawMode)
{
      Point_t   CurrPoint;

      CurrPoint.xCoord = X_VERT_AXIS+1; /* don't write on top of the vertical
axis */
      switch (DrawMode)
      {
            case DRAW_DIFFERENTIAL:
                  CurrPoint.yCoord = HC_ptr->PositionDifferential;
                  break;
            case DRAW_ZOOM:
            case DRAW_ABSOLUTE:
                  CurrPoint.yCoord = HC_ptr->PositionAbsolute;
                  break;
            default: /* what the heck happened here?!? */
                  CurrPoint.yCoord = HC_ptr->PositionAbsolute;
                  break;
      }
      SetDrawingCursor(&CurrPoint);
      CurrPoint.xCoord += TOTAL_SPREETA_PIXELS-1;
      CurrPoint.yCoord += HC_ptr->Width - 1;
      Box(&DefaultRenderingContext, &CurrPoint, 1, COLOR_WHITE);

}
/************************************************************
 *
 * CheckEraseAtCursor()
 *
 * Check for graph draw overlay with cursor and execute a draw only if
 * interference is detected;  If the cursor and the pixel are exactly
 * on top of each other, the cursor is showing and no erase in necessary.
 *
 * Inputs:
 *     PixelCount - the pixel location to be erased from the graph (2 high)
 *     CursorPosition - the horizontal height of the cursor
 *     CursorColor - the erase color for interference...
 *
 * returns:
 *     ERASURE_COMPLETE if interference was detected
 *        NO_INTERFERENCE  otherwise
 *
 ******************************************************************/
int CheckEraseAtCursor(int PixelCount, int CursorPosition, int CursorColor)
{
      switch (gPreviousScreenValue[PixelCount] - CursorPosition)
      {
            case 0:      /* don't do any work */
                  break;
            case -1:  /* erase 1/2 above red line and 1/2 on red line */

      WritePixel(&DefaultRenderingContext,X_VERT_AXIS+1+PixelCount,gPreviousS
creenValue[PixelCount],COLOR_WHITE);

      WritePixel(&DefaultRenderingContext,X_VERT_AXIS+1+PixelCount,gPreviousS
creenValue[PixelCount]+1,CursorColor);
                  break;
            case 1:  /* erase 1/2 below red line and 1/2 on red line */

      WritePixel(&DefaultRenderingContext,X_VERT_AXIS+1+PixelCount,gPreviousS
creenValue[PixelCount],CursorColor);

      WritePixel(&DefaultRenderingContext,X_VERT_AXIS+1+PixelCount,gPreviousS
creenValue[PixelCount]+1,COLOR_WHITE);
                  break;
            default:
                   return(NO_INTERFERENCE);
                   /* break; */  /* compiler warning if break is left in
after return*/
                                 /* actually, it's a pretty impressive
compiler, eh? */
       }

       return(ERASURE_COMPLETE);
}
/**************************************************************
 *
 * DrawAxes()
 *
 * Just gather the draws into one convenient spot so we don't have to
 * maintain them in more than one spot...
 *
 ***************************************************************/
void DrawAxes(void)
{
       int i;
       char pScreenBuff[20];  /* room for the sprintf() text */

       WriteLine(&DefaultRenderingContext,   /* Vertical Axis */

X_VERT_AXIS,Y_TOP_VERT_LABEL,X_VERT_AXIS,Y_HORZ_AXIS,COLOR_BLACK);
       WriteLine(&DefaultRenderingContext,  /* Horizontal Axis */

X_VERT_AXIS,Y_HORZ_AXIS,X_VERT_AXIS+TOTAL_SPREETA_PIXELS,Y_HORZ_AXIS,COLOR_BLA
CK);
       WriteLine(&DefaultRenderingContext,  /* top Tick */
                  X_VERT_AXIS-
2,TOP_GRAPH_BOUNDARY,X_VERT_AXIS+2,TOP_GRAPH_BOUNDARY,COLOR_BLACK);

       for (i=0; i<=3; i++)
       {
        sprintf(pScreenBuff, "%-04.3f",RIU_MIN + ((RIU_MAX - RIU_MIN) *
(float)i / 3));

       PrintTextToScreen(Y_HORZ_AXIS_VALUES,(int)((X_VERT_AXIS+(float)i/3*TOTA
L_SPREETA_PIXELS)-14),(string)pScreenBuff);
              WriteLine(&DefaultRenderingContext,

(int)(X_VERT_AXIS+(float)i/3*TOTAL_SPREETA_PIXELS),Y_HORZ_AXIS-2,

(int)(X_VERT_AXIS+(float)i/3*TOTAL_SPREETA_PIXELS),Y_HORZ_AXIS+2,
                        COLOR_BLACK);
       }


       PrintTextToScreen(Y_TOP_VERT_LABEL-TEXT_LINE_SPACING-
1,X_VERT_LABEL,VERTICAL_AXIS_TEXT);
       PrintTextToScreen(Y_HORZ_AXIS_LABEL,X_HORZ_AXIS_LABEL,HORIZONTAL_AXIS_T
EXT);
       PrintTextToScreen(Y_SAMPLE_COUNT_LABEL,X_SAMPLE_LABEL,"Sample Count:");
       PrintTextToScreen(Y_AVG_LABEL,X_AVG_MIN_MAX_LABEL,"Avg Val");
       PrintTextToScreen(Y_AVG_LABEL+TEXT_LINE_SPACING,X_AVG_MIN_MAX_VALUE,"0.
000 V");
       PrintTextToScreen(Y_MIN_LABEL,X_AVG_MIN_MAX_LABEL,"Min Val");
       PrintTextToScreen(Y_MIN_LABEL+TEXT_LINE_SPACING,X_AVG_MIN_MAX_VALUE,"0.
000 V");
       PrintTextToScreen(Y_MIN_LABEL+TEXT_LINE_SPACING*2,X_AVG_MIN_MAX_VALUE,"
0.000 RIU");
       PrintTextToScreen(Y_MAX_LABEL,X_AVG_MIN_MAX_LABEL,"Max Val");
       PrintTextToScreen(Y_MAX_LABEL+TEXT_LINE_SPACING,X_AVG_MIN_MAX_VALUE,"0.
000 V");
```

```
      PrintTextToScreen(Y_MAX_LABEL+TEXT_LINE_SPACING*2,X_AVG_MIN_MAX_VALUE,"
0.000 RIU");

}

/*************************************************************
 *
 * display_absolute_screen()
 *
 * builds the screen for the "absolute data" mode
 *
 * with title, x&y axes, status bar, etc...
 *
 *  5  =>                   Draw Mode
 *  10 => 3.00V \
 *             \                                    Avg Val
 *              \                                         0
 *   50 => 2.00V \******************
 *             \                                    Min Val
 *              \                                         0
 *   90 => 1.00V \
 *             \                                    Max Val
 *              \                                         0
 * 130 => 0.00V \______________________________
 * 140 =>        Min      Incident Angle   Max
 * 150 => Sample Count: 0
 *
 *        ^     ^            ^                 ^       ^
 *        10    30           80                          158       170
 *************************************************************/
void display_absolute_screen(void)
{
      #ifdef NOISY
      printf("entering display_absolute_screen()\n");
      #endif

      ClearScreen(&DefaultRenderingContext,COLOR_WHITE);
      DrawAxes();
      WriteLine(&DefaultRenderingContext,  /* 2nd Tick */
                 X_VERT_AXIS-
2,Y_2ND_VERT_LABEL+TEXT_HEIGHT,X_VERT_AXIS+2,Y_2ND_VERT_LABEL+TEXT_HEIGHT,COLO
R_BLACK);
      WriteLine(&DefaultRenderingContext,  /* 3rd Tick */
                 X_VERT_AXIS-
2,Y_3RD_VERT_LABEL+TEXT_HEIGHT,X_VERT_AXIS+2,Y_3RD_VERT_LABEL+TEXT_HEIGHT,COLO
R_BLACK);
      PrintTextToScreen(Y_TITLE,X_TITLE,"Absolute Value Scale");
      PrintTextToScreen(Y_TOP_VERT_LABEL+(TEXT_HEIGHT/2),X_VERT_LABEL,"3.00V"
);
      PrintTextToScreen(Y_2ND_VERT_LABEL+(TEXT_HEIGHT/2),X_VERT_LABEL,"2.00V"
);
      PrintTextToScreen(Y_3RD_VERT_LABEL+(TEXT_HEIGHT/2),X_VERT_LABEL,"1.00V"
);
      PrintTextToScreen(Y_BOT_VERT_LABEL+2,X_VERT_LABEL,"0.00V");

      DrawHorizontalCursor(&gHorizontalCursor1,DRAW_ABSOLUTE);
      DrawHorizontalCursor(&gHorizontalCursor2,DRAW_ABSOLUTE);
      UpdateDateOnScreen();
      UpdateTimeOnScreen();
}


/*************************************************************
 *
 * display_differential_screen()
 *
 * builds the screen for the "differential data" mode
 *
 * with title, x&y axes, status bar, etc...
 *
 *  5  =>                   Draw Mode
 *  10 =>  +XV \
 *             \                                                 Avg Val
 *              \                                                      0
 *   50 =>      \
 *          0V \******************                          Min Val
 *              \                                                      0
 *   90 =>      \
 *             \                                                 Max Val
 *              \                                                      0
 * 130 =>  -XV \______________________________
 * 140 =>       Min      Incident Angle   Max
 * 150 => Sample Count: 0
 *
 *        ^    ^            ^                  ^       ^
 *        10   30          80                       158     170
 *****************************************************************/
void display_differential_screen(void)
{
      char pScreenBuff[50]; /* used for printing the vertical axis labels to
the screen */

      #ifdef NOISY
      printf("entering display_differential_screen()\n");
      #endif

      ClearScreen(&DefaultRenderingContext,COLOR_WHITE);
      DrawAxes();
      PrintTextToScreen(Y_TITLE,X_TITLE,"Differential Value Scale");

                            gScreenMaxValue = (Y_BOT_VERT_LABEL -
Y_TOP_VERT_LABEL)/2
                                                                    *
(int)(pow(2,(double)gDifferentialTicksPerPixelPower));
                            gScreenMinValue = -gScreenMaxValue;


      /* calculate the values to print to the screen for the Vertical-Axis */
    sprintf(pScreenBuff, "%-04.2fV",(Y_BOT_VERT_LABEL - Y_TOP_VERT_LABEL)/2
                                                                    *
(int)(pow(2,(double)gDifferentialTicksPerPixelPower))
                                                                    *
((float)3/SPREETA_ADC_3V_VALUE));
      PrintTextToScreen(Y_TOP_VERT_LABEL+(TEXT_HEIGHT/2),X_VERT_LABEL,(string
)pScreenBuff);

    sprintf(pScreenBuff, "%-04.2fV",0.0);
      PrintTextToScreen(Y_TOP_VERT_LABEL+(TEXT_HEIGHT/2) + (Y_BOT_VERT_LABEL
- Y_TOP_VERT_LABEL)/2,X_VERT_LABEL,(string)pScreenBuff);

    sprintf(pScreenBuff, "%-04.2fV",-(Y_BOT_VERT_LABEL - Y_TOP_VERT_LABEL)/2
                                                                    *
(int)(pow(2,(double)gDifferentialTicksPerPixelPower))
                                                                    *
((float)3/SPREETA_ADC_3V_VALUE));
      PrintTextToScreen(Y_BOT_VERT_LABEL+2,X_VERT_LABEL-(TEXT_WIDTH -
2),(string)pScreenBuff);

      DrawHorizontalCursor(&gHorizontalCursor1,DRAW_DIFFERENTIAL);
      DrawHorizontalCursor(&gHorizontalCursor2,DRAW_DIFFERENTIAL);

      /* draw the red line at the 0 differential point */
      DrawHorizontalCursor(&gDifferentialCenterLine,DRAW_DIFFERENTIAL);
      WriteLine(&DefaultRenderingContext,  /* Midle Tick */
                 X_VERT_AXIS-2,gDifferentialCenterLine.PositionDifferential,

X_VERT_AXIS+2,gDifferentialCenterLine.PositionDifferential,COLOR_BLACK);
    UpdateDateOnScreen();
    UpdateTimeOnScreen();
}

/**************************************************************
 *
 * display_zoom_screen()
 *
 * builds the screen for the "zoom data" mode
 *
 * with title, x&y axes, status bar, etc...
 *
 *  5  =>                   Draw Mode
 *  10 => MAXV \
 *             \                                              Avg Val
 *              \                                                  0
 *   50 => MIDV \******************
 *             \                                              Min Val
 *              \                                                  0
 *   90 => MIDV \
 *             \                                              Max Val
 *              \                                                  0
 * 130 => MINV \______________________________
 * 140 =>       Min      Incident Angle   Max
 * 150 => Sample Count: 0
 *
 *        ^    ^            ^                  ^       ^
 *        10   30           80                      158     170
 ***************************************************************/
void display_zoom_screen(void)
{

      char pScreenBuff[50]; /* used for printing the vertical axis labels to
the screen */
      int i;  /* poorly named loop counter habit left over from old die-hard
school examples */

      #ifdef NOISY
      printf("entering display_zoom_screen()\n");
      #endif

      ClearScreen(&DefaultRenderingContext,COLOR_WHITE);
      DrawAxes();
      WriteLine(&DefaultRenderingContext,  /* 2nd Tick */
                 X_VERT_AXIS-
2,Y_2ND_VERT_LABEL+TEXT_HEIGHT,X_VERT_AXIS+2,Y_2ND_VERT_LABEL+TEXT_HEIGHT,COLO
R_BLACK);
      WriteLine(&DefaultRenderingContext,  /* 3rd Tick */
                 X_VERT_AXIS-
2,Y_3RD_VERT_LABEL+TEXT_HEIGHT,X_VERT_AXIS+2,Y_3RD_VERT_LABEL+TEXT_HEIGHT,COLO
R_BLACK);
      PrintTextToScreen(Y_TITLE,X_TITLE,"Zoom Value Scale");

      /* calculate the values to print to the screen for the Vertical-Axis */
      for (i=0; i<=3; i++)
      {
       sprintf(pScreenBuff, "%-04.2fV",((gScreenMinValue + (gScreenMaxValue -
gScreenMinValue) * (float)i / 3)) * ((float)3/SPREETA_ADC_3V_VALUE));
            PrintTextToScreen(Y_BOT_VERT_LABEL-
(Y_VERT_LABEL_SPACING*i),X_VERT_LABEL,(string)pScreenBuff);
      }

      DrawHorizontalCursor(&gHorizontalCursor1,DRAW_ZOOM);
      DrawHorizontalCursor(&gHorizontalCursor2,DRAW_ZOOM);
      UpdateDateOnScreen();
      UpdateTimeOnScreen();
} /* end: display_zoom_screen */

/**************************************************************
 *
 * OutlineBox()
 *
 * Draws a un-filled box around the perimiter of a boxed area
 *
 * Inputs:
 *    TopLeft: the (x,y) of the top left of the box
 *    BottomRight: the (x,y) of the bottom right of the box
 *
 ********************************************************************/
OutlineBox(Point_t TopLeft, Point_t BottomRight, int LineColor)
{
      WriteLine(&DefaultRenderingContext,  /* Top */

TopLeft.xCoord,TopLeft.yCoord,BottomRight.xCoord,TopLeft.yCoord,LineColor);
      WriteLine(&DefaultRenderingContext,  /* Bottom */

TopLeft.xCoord,BottomRight.yCoord,BottomRight.xCoord,BottomRight.yCoord,LineCo
lor);
      WriteLine(&DefaultRenderingContext,  /* Left */

TopLeft.xCoord,TopLeft.yCoord,TopLeft.xCoord,BottomRight.yCoord,LineColor);
      WriteLine(&DefaultRenderingContext,  /* Right */

BottomRight.xCoord,TopLeft.yCoord,BottomRight.xCoord,BottomRight.yCoord,LineCo
lor);

} /* end OutlineBox */


/**************************************************************
 *
 * DrawPopUp()
 *
 * builds the pop-up for user data input
 *
 * Input:
 *   TitleText - the text to center in the title of the PopUp
 *
 * Output:
 *   TextPoint - the (x,y) point of the cursor
 *
 *
 *  30 =>       ______________________________
 *             \                                  \
 *             \       Title Of PopUp         \
 *              \______________________________\
 *             \*                              \
 *             \                               \
 *              \                                 \
 *              \                                 \
 *             \                               \
 *              \                                 \
 * 129 =>      \______________________________\
 *
 *             ^                             ^
 *             30                       209
 ********************************************************************/
Point_t DrawPopUp(char *TitleText)
{
	Point_t TopLeft, BottomRight;

	#ifdef NOISY
	printf("entering DrawPopUp\n");
	#endif

	/* Draw the main outer box */
	TopLeft.xCoord = POPUP_TOP_X;
	TopLeft.yCoord = POPUP_TOP_Y;
	SetDrawingCursor(&TopLeft);
	BottomRight.xCoord = POPUP_BOTTOM_X;
	BottomRight.yCoord = POPUP_BOTTOM_Y;
	Box(&DefaultRenderingContext, &BottomRight, 1, COLOR_LIGHT_RED);
	OutlineBox(TopLeft, BottomRight, COLOR_BLACK);

	/* Draw the title Box */
	TopLeft.xCoord = POPUP_TOP_X+POPUP_BOARDER_THICKNESS;
	TopLeft.yCoord = POPUP_TOP_Y+POPUP_BOARDER_THICKNESS;
	SetDrawingCursor(&TopLeft);
	BottomRight.xCoord = POPUP_BOTTOM_X - POPUP_BOARDER_THICKNESS;
	BottomRight.yCoord = TopLeft.yCoord + (TEXT_HEIGHT +
SPACE_BETWEEN_TEXT_LINES*2) + 2;
	Box(&DefaultRenderingContext, &BottomRight, 1, COLOR_WHITE);
	OutlineBox(TopLeft, BottomRight,COLOR_BLACK);

	/* Center the text in the PopUp title */
	TopLeft.xCoord += (POPUP_BOTTOM_X - POPUP_TOP_X)/2 -
StringWidth((string)TitleText,MINIME_TEXT)/2;
	TopLeft.yCoord += SPACE_BETWEEN_TEXT_LINES +2;
	PrintTextToScreen(TopLeft.yCoord, TopLeft.xCoord, (string)TitleText);

	/* Draw the User Input Box */
	TopLeft.xCoord = POPUP_TOP_X+POPUP_BOARDER_THICKNESS;
	TopLeft.yCoord += TEXT_HEIGHT + SPACE_BETWEEN_TEXT_LINES +
POPUP_BOARDER_THICKNESS;
	SetDrawingCursor(&TopLeft);
	BottomRight.xCoord = POPUP_BOTTOM_X - POPUP_BOARDER_THICKNESS;
	BottomRight.yCoord = POPUP_BOTTOM_Y - POPUP_BOARDER_THICKNESS;
	Box(&DefaultRenderingContext, &BottomRight, 1, COLOR_WHITE);
	OutlineBox(TopLeft, BottomRight, COLOR_BLACK);


	/* Initialize the cursor in the User Input box */
	TopLeft.xCoord += CURSOR_WIDTH;
	TopLeft.yCoord += CURSOR_HEIGHT;
	SetDrawingCursor(&TopLeft);
	BottomRight.xCoord = TopLeft.xCoord + CURSOR_WIDTH;
	BottomRight.yCoord = TopLeft.yCoord + CURSOR_HEIGHT;
	Box(&DefaultRenderingContext, &BottomRight, 1, COLOR_BLACK);

	/* Initialize the Text Cursor */
	SetTextCursor(&TopLeft);
	return(TopLeft);

} /* end: DrawPopUp */

/**************************************************************************
 * SampleSpreeta()
 *
 * captures all the spreeta data for a single run uf samples.  This is a
 * very optimized loop to keep the sample rate as:
 * a) fast as possible (min Sample Frequency is 5KHz)
 * b) consistant as possible (data values delivered by the pixel array
 *    are a function of sample rate
 *
 * the input paramater tells us if we're just priming the Spreeta or if
 * we're truly capturing the data:
```

```
 * CAPTURE_PIXELS: store data read into the captured values array
 * BEGIN_INTEGRATION: run through a sampel cycle but thow away the data

 * Note: if we give a runt ChipSelect to the Maxim part, even if we
 *       aren't reading the device, it will give us garbage data
 *       on the next real conversion...
 *
 *****************************************************************/
 void SampleSpreeta(int CaptureMode)
 {

       volatile int DummyData;  /* used to throw away SPI data we don't care
about */
       int          PixelCount; /* Outer Loop control for reading the entire
array */
       int                 BitsToShift; /* only need 18 clocks if
BEGIN_INTEGRATION */
       u_int16             SpreetaBoth;
       u_int16      SpreetaStart;
       u_int16      SpreetaClock;
       u_int16      MaximCS;
       u_int16      ShiftRegAllOff;


       /* if we're beginning integration, turn on the LED simultaniously with
the Start
          bit to the spreeta.  Then give 18 clocks to end the reset period and
begin
          the integration period.  Thus the LED must be turned on and stay on
          until the next capture */
       if (CaptureMode == BEGIN_INTEGRATION)
       {
             /* BitsToShift    = SPREETA_RESET_CLOCKS; */
             BitsToShift    = TOTAL_SPREETA_PIXELS;
             SpreetaBoth    = SPREETA_BOTH | SPREETA_LED_ON;
             SpreetaStart   = SPREETA_START | SPREETA_LED_ON;
             SpreetaClock   = SPREETA_CLOCK | SPREETA_LED_ON;
          MaximCS        = MAXIM_CS | SPREETA_LED_ON;
             ShiftRegAllOff = SHIFT_REG_ALL_OFF | SPREETA_LED_ON;
       }
       else
       {
             BitsToShift = TOTAL_SPREETA_PIXELS;
             SpreetaBoth = SPREETA_BOTH;
             SpreetaStart = SPREETA_START;
             SpreetaClock = SPREETA_CLOCK;
             MaximCS = MAXIM_CS;
             ShiftRegAllOff = SHIFT_REG_ALL_OFF;
       }

       SPI_SET_SPREETA;
       /* set Spreeta's Start Bit */
       *SPI_TRANSMIT = SpreetaStart;
       SPI_EXCHANGE;       /* shift out the Start data to the shift register */
       WAIT_EXCHANGE;
       DummyData = *SPI_RECEIVE; /* empty out the receive buffer */

       /* Set Spreeta's Clock while maintaing Start */
       *SPI_TRANSMIT = SpreetaBoth;
       SPI_EXCHANGE;       /* shift out the Clock and Start data to the shift
register */
       WAIT_EXCHANGE;
       DummyData = *SPI_RECEIVE; /* empty out the receive buffer */

       /* Start the sample on the A/D converter
        * note that the Clock just went high and the Start bit is high
        * the Spreeta takes 350ns to drive it's output:
```

```
        *      the DragonBall runs at 33MHz => a clock cycle is 30ns
        *      the memory is 1 wait-state so every bus cycle is 4x30ns = 120ns
        *      the bus is only 16 bits wide so every instruction is at least 2
bus cycles = 240ns
        *      thus 2 instructions is plenty of time, no delay loop needed
        *
        *  as measured on the logic analyzer, we have ~800ns between Spreeta
Clock and Maxim CS*/

    *SPI_TRANSMIT = MaximCS;
      SPI_EXCHANGE;       /* Drive the Maxim CS low */
      WAIT_EXCHANGE;

      for (PixelCount=0; PixelCount<BitsToShift; PixelCount++)
      {
             /* want to maximize delay between MAXIM_CS and Reading the Maxim
data
                becaue there's a spec. of 7.1uSec for the sample capture in
the IC
                - the loop control will add the delay no matter if we're just
entering
                  it or if we're repeating */
             DummyData = *SPI_RECEIVE; /* empty out the receive buffer here to
increase delay */

             SPI_SET_MAXIM; /* set up for a 13-clock transfer */
             *SPI_TRANSMIT = SpreetaClock; /* load the FIFO to set the Spreeta
clock at the end of reading the Maxim A/D */
             SPI_EXCHANGE;       /* read the 12 data bits into the status
register */
             WAIT_EXCHANGE;

             if (CaptureMode == CAPTURE_PIXELS)
                 gCurrentSpreetaValue[PixelCount] = *SPI_RECEIVE & ~(0xF000);
/* clear off the top junk bit(s) */
             else /* BEGIN_INTEGRATION */
                 DummyData = *SPI_RECEIVE; /* throw away the data */

             SPI_SET_SPREETA;
             *SPI_TRANSMIT = MaximCS;
             SPI_EXCHANGE;       /* Drive the Maxim CS low */
             /* WAIT_EXCHANGE */ /* no need for a delay, the loop control will
instert plenty */
             #ifdef VERY_NOISY
             help!! If interrupts are disabled, this isn't goona work...
             printf("Spreeta Value:
0x%03x\n",gCurrentSpreetaValue[PixelCount]);
             #endif

      } /* end outer for loop */

      /* upon exit from the loop, we just asserted the MAXIM chip select, so
we need
         to turn it back off and initalize the system back to everything
turned off */
      DummyData = *SPI_RECEIVE; /* empty out the receive buffer for the MAXIM
CS Assertion */
      *SPI_TRANSMIT = ShiftRegAllOff; /* turn off the Maxim CS; leave LED on
if BEGIN_INTEGRATION */
      SPI_EXCHANGE;       /* drive the 4 data bits into the shift register */
      WAIT_EXCHANGE;
      DummyData = *SPI_RECEIVE;

} /* end SampleSpreeta */

/***************************************************************************
 * SmoothingAlgorithm()
 *
 * passes back an averaged value.
 *
 * in this instantiation it's hard-coded to be one
 * simple function and optmized to not involve division:
 *
 * SmoothValue = ((2_to_the_left + 1_to_the_left + 1_to_the_right +
2_to_the_right)/4
 *             + pixelValue)/2
 *
 * the algorithm can smooth 5 values, giving heavier weighting to the
 * value at the actual location and weighting to the values left & right.
 ***********************************************************/
int SmoothingAlgorithm(int pixel_count)
{
      int SmoothedSpreetaValue; /* return value for this function */

      /* skip pixels 0, 1, 126, 127 */
      if ((pixel_count >= 2) && (pixel_count < TOTAL_SPREETA_PIXELS - 2))
      {
            /* average the 2 to the left + the 2 to the right */
            SmoothedSpreetaValue = gCurrentSpreetaValue[pixel_count-2]
                                 + gCurrentSpreetaValue[pixel_count-1]
                                 + gCurrentSpreetaValue[pixel_count+1]
                                 + gCurrentSpreetaValue[pixel_count+2];

            #ifdef NOISY_SMOOTHING
                  printf("Smooth[%d]: %04x + %04x + %04x + %04x = %04x\n",
                         pixel_count,
                         gCurrentSpreetaValue[pixel_count-2],
                         gCurrentSpreetaValue[pixel_count-1],
                         gCurrentSpreetaValue[pixel_count+1],
                         gCurrentSpreetaValue[pixel_count+2],
                                 SmoothedSpreetaValue);
            #endif

            SmoothedSpreetaValue >>= 2; /* divide by 4 */
            #ifdef NOISY_SMOOTHING
                  printf("Smooth: Div_4: %04x\n",SmoothedSpreetaValue);
            #endif

            /* now average the center value with the averaged 2 left & 2
right */

            #ifdef NOISY_SMOOTHING
                  printf("Smooth: %04x + %04x =",SmoothedSpreetaValue,
gCurrentSpreetaValue[pixel_count]);
            #endif
            SmoothedSpreetaValue += gCurrentSpreetaValue[pixel_count];
            #ifdef NOISY_SMOOTHING
                  printf(" %04x\n",SmoothedSpreetaValue);
            #endif
            SmoothedSpreetaValue >>= 1; /* divide by 2 */
            #ifdef NOISY_SMOOTHING
                  printf("Smooth: Div_2: %04x\n",SmoothedSpreetaValue);
            #endif
      }
      else /* we're at the ends of the sample, just use the raw value */
      {
            SmoothedSpreetaValue = gCurrentSpreetaValue[pixel_count];
            #ifdef NOISY
                  printf("Smooth[%d] raw value: %04x\n",
                         pixel_count, SmoothedSpreetaValue);
            #endif
      }

      return(SmoothedSpreetaValue);
}


/************************************************************************
 * ProcessPixelValue()
 *
 * passes back the pixel value to draw to the screen.
 *
 * works in two modes:
 *
 * a) in differential mode, it subtracts the previous value from the last
falue
 * b) in aboslute mode, it simply passes back the pixel value
 *
 * this route also maintains the "previous data" buffer
 * so it can calculate the differential (if neccessairy)
 *
 *
 * VREF = 3.0V
 *
 * 3.0V => FFF => 4095
 * 2.0V => AAA => 2730
 * conversion to decimal: 3 x spreet_pixel_value / 4095
 *
 * conversion to absolute screen location:
 * Y_TOP_VERT_LABEL + (4095 - gCurrentSpreetaValue)/((4095 -
2730)/Y_VERT_LABEL_SPACING)
 *
 ***************************************************************/
int ProcessPixelValue(int pixel_count, int algorithm_mode)
{
      int screen_pixel_value=0;   /* algorithm-specific value returned for
drawing to the screen */
      int SmoothedSpreetaValue = 0;  /* smooth the value before plotting */

      switch (algorithm_mode)
      {
            case DRAW_ABSOLUTE:
            case DRAW_ZOOM:
                  /*
                   *     Y_AXIS + (SPREETA_DATA * SPREETA=>SCREEN
Conversion)
                   */

                  screen_pixel_value = (int)((BOTTOM_GRAPH_BOUNDARY)  /* the
zero axis minus the screen offset (minus moves up) */
                                              -
(int)((int)(gCurrentSpreetaValue[pixel_count] - gScreenMinValue) *
gConvertADCToScreen)
                                             );

                  break;
            case DRAW_DIFFERENTIAL:

                /*
                   *   Draws a flat line in the middle of the screen,
waiting for a pixel to move
                   *   off of that flat line, either up or down, depending
on whether the light has increased
                   *   or decreased at that particular angle...
                   *
                   *   Y_AXIS + (OLD_SPREETA_DATA - NEW_SPREETA_DATA +
MIDSCREEN_Offset) * (SPREETA=>SCREEN Conversion)
                   */

                  SmoothedSpreetaValue = SmoothingAlgorithm(pixel_count);
```

```
              screen_pixel_value = TOP_GRAPH_BOUNDARY + (Y_BOT_VERT_LABEL -
Y_TOP_VERT_LABEL)/2  /* midpoint of screen */
                                                     + (
(gBaselinedSpreetaValue[pixel_count] - SmoothedSpreetaValue)
                                                            >>
gDifferentialTicksPerPixelPower   /* right shift avoids floating-point */
                                                       )
                                                       ;
                 break;
          default:
                    screen_pixel_value = TOP_GRAPH_BOUNDARY;
                    #ifdef NOISY
                    printf("in ProcessPixelValue: error!! invalid
algorithm_mode: %d\n",algorithm_mode);
                    #endif
                    break;
      } /* end switch algorithm_mode */

      #ifdef VERY_NOISY
      printf("Screen value: %d\n",screen_pixel_value);
      #endif

      /* keep the values on the screen... */
      if (screen_pixel_value > Y_HORZ_AXIS-2)
            screen_pixel_value = Y_HORZ_AXIS-2;

      if (screen_pixel_value < TOP_GRAPH_BOUNDARY)
            screen_pixel_value = TOP_GRAPH_BOUNDARY;

      return(screen_pixel_value);
}

void CompareMaxValue(int CompareValue, int PixelCount)
{
      if (PixelCount == 0)
      {
          gSpreetaMaxValue = CompareValue;
            gSpreetaMaxValueLocation = PixelCount;
      }
      else
      {
            if (CompareValue > gSpreetaMaxValue)
            {
                gSpreetaMaxValue = CompareValue;
                  gSpreetaMaxValueLocation = PixelCount;
            }
      }
}


void CompareMinValue(int CompareValue, int PixelCount)
{
      if (PixelCount == 0)
      {
            gSpreetaMinValue = CompareValue;
            gSpreetaMinValueLocation = PixelCount;
      }
      else
      {
            if (CompareValue < gSpreetaMinValue)
            {
                gSpreetaMinValue = CompareValue;
                  gSpreetaMinValueLocation = PixelCount;
            }
      }
}
```

```
/**************************************************************************
 * BuildSMSMessage
 *
 * Upon detection of a threshold violation, this routine will spawn
 * the wireless notification to the user
 *
 ****************************************************************/
fisID_t BuildSMSMessage(char *AddressList, char *MessageText)
{
      methodParam_t             methodParam;
      mailObject_t        obj;
      int erc;
      fisID_t fisID;
      u_int8   DummyReadBuff[8]; /* read buffer to search for unused mailobj
*/
      u_int32  BuffLen;

      #ifdef NOISY
      printf("[SPREETA] entering BuildSMSMessage with (%s) and (%s)\n",
             AddressList, MessageText);
      #endif

      /* Initialize the mail object */
      erc = mailObject(&obj, MSG_CREATE_OBJECT, &methodParam);

      if (!erc)
      {
             erc = mailObject(&obj, MSG_SETUARDEFAULTS, &methodParam);

             methodParam.Param = 1;
             erc = mailObject(&obj, SET_ADRLISTCNT, &methodParam);

             /* Set the carrier id to Reflex 50 */
             methodParam.Param = ADR_TYPE_EMAIL;
             methodParam.ItemNum = 1;
             erc = mailObject(&obj, SET_ADRTYPE, &methodParam);

             methodParam.ParamStr = AddressList;
             methodParam.ItemNum = 1;
             erc = mailObject(&obj, SET_ADRVAL, &methodParam);

             methodParam.Param = strlen(MessageText);
             erc = mailObject(&obj, SET_CONTENTLEN, &methodParam);

             methodParam.ParamStr = (char *) MessageText;
             methodParam.ItemNum = 1;
             erc = mailObject(&obj, SET_CONTENTVAL, &methodParam);

             /* create mailobjects in a seperate place to keep track of them
and distroy them */
             fisID = MAILOBJ_FIS_ID;
             BuffLen = sizeof(DummyReadBuff);

             /* search for a free ID to store the bitmap */
             while (FISRead(INCOMING_MAIL_TYPE, fisID, DummyReadBuff,&BuffLen)
!= E_ITEM_NOT_FOUND)
             {
                   #ifdef NOISY
                   printf("[SPREETA] Skipping over MailObj ID: %d\n",fisID);
                   #endif
                   fisID++;
             }

             methodParam.Param = COMPOSE_FIS_ID(INCOMING_MAIL_TYPE,fisID);
             obj.objectHdr.mailbinID = methodParam.Param;
             erc = mailObject(&obj, MSG_SAVE, &methodParam);
```

```
            fisID = methodParam.Param;

#ifdef NOISY
            if (!erc)
                  printf("Just created item %d\n", fisID);
            else
                  printf("MSG_SAVE returned %d\n", erc);
#endif
      erc = mailObject(&obj, MSG_DESTROY_OBJECT, &methodParam);
      }
      return fisID;
}

/***************************************************************************
 * SpawnNotification
 *
 * Upon detection of a threshold violation, this routine will spawn
 * the wireless notification to the user
 *
 ****************************************************************/
void SpawnNotification(void)
{

      eventmsg SMSEventMsg;
      process_id SpreetaPID;
      process_id MDSPID;
      struct SMSNotification *SearchSMSList;
      Point_t CurrPoint;
      char *PrintString;

      #ifdef NOISY
      printf("entering SpawnNotification\n");
      #endif

      for ( SearchSMSList = gpSMSNotificationList;
           (SearchSMSList != NULL) && (SearchSMSList->Sent == TRUE);
            SearchSMSList = SearchSMSList->Next)
              {}

      if (SearchSMSList != NULL)
      {
            if (gNotificationStatus   == NOTIFICATION_ENABLED)
            {
                  CurrPoint = DrawPopUp("Sending Notificaiton(s)");
                  EraseTextCursor();
             }

            GetTextCursor(&CurrPoint);
            PrintTextToScreen(CurrPoint.yCoord, CurrPoint.xCoord, "Sending
alert to address:");
            CurrPoint.yCoord += TEXT_LINE_SPACING;
            PrintString = malloc(strlen(SearchSMSList->Address)+2);
            strcpy(PrintString,"  ");
            strcat(PrintString,SearchSMSList->Address);
            PrintTextToScreen(CurrPoint.yCoord, CurrPoint.xCoord,
(string)PrintString);
            CurrPoint.yCoord += TEXT_LINE_SPACING;
            SetTextCursor(&CurrPoint);

            gNotificationStatus = NOTIFICATION_IN_PROGRESS;
            name2pid("spreeta", &SpreetaPID);
            name2pid("mmds", &MDSPID);

            SMSEventMsg.FromPid = SpreetaPID;
            SMSEventMsg.EventCode = EVENT_NEW_OUTMAIL;
            SMSEventMsg.asTask = 0;
```

```
             SMSEventMsg.Param1 = COMPOSE_FIS_ID(INCOMING_MAIL_TYPE,
SearchSMSList->SMS_ID); /* MailbinID is in param1 */
             SMSEventMsg.Param2 = CARRIER_REFLEX_50;   /* Carrier ID is in
param2 */

             #ifdef NOISY
             printf("Sending EVENT_NEW_OUTMAIL (0x%08x) event to
MDS\n",EVENT_NEW_OUTMAIL);
             #endif
             SendMsg(MDSPID, &SMSEventMsg);
             SearchSMSList->Sent = TRUE;
      }
      else /* clean up the linked list */
      {
             while (gpSMSNotificationList != NULL)
             {
                   SearchSMSList = gpSMSNotificationList;
                   gpSMSNotificationList = gpSMSNotificationList->Next;
                   free(SearchSMSList->Address);
                   free(SearchSMSList);
             }
             GetTextCursor(&CurrPoint);
             /* very dangerous, if we get out of sequence, we're going to look
rediculous right on the screen */
             PrintTextToScreen(CurrPoint.yCoord, CurrPoint.xCoord, "Press
<esc> to continue...");
             CurrPoint.yCoord += TEXT_LINE_SPACING;
             SetTextCursor(&CurrPoint);
             gNotificationStatus = NOTIFICATION_DORMANT;
      }
}
/**************************************************************************
 * ProcessSystemEvent
 *
 * Input from the event handler that isn't a keypress.  Presumably
 * it's from MDS in repsonse to the SMS message...
 *
 ****************************************************************/
void ProcessSystemEvent(eventmsg InputEvent)
{

      Point_t CurrPoint;

      #ifdef NOISY
      printf("[SPREETA] entering ProcessSystemEvent with event:
0x%x\n",InputEvent.EventCode);
      #endif

      switch (InputEvent.EventCode)
      {
             case EVENT_PROFILE_CHANGE:
                   #ifdef NOISY
                   printf("[SPREETA] SystemEvent EVENT_PROFILE_CHANGE: you're
in PDA Mode?\n");
                   #endif
                   break;

             case EVENT_SIGNAL_STRENGTH:
                   #ifdef NOISY
                   printf("[SPREETA] SystemEvent EVENT_SIGNAL_STRENGTH:
ignoring...\n");
                   #endif
                   break;

             case EVENT_BATTERYDOOR_CLOSE:
                   #ifdef NOISY
```

```
                    printf("[SPREETA] SystemEvent EVENT_BATTERYDOOR_CLOSE;
SubEvent 0x%x:\n",InputEvent.Param1);
                    #endif
                            switch (InputEvent.Param1)
                            {
                                    default:
                                    break;
                            }
                    break;
            case EVENT_MAIL_STATUS:
                            #ifdef NOISY
                            printf("[SPREETA] SystemEvent EVENT_MAIL_STATUS;
Param1 0x%x; Param2 0x%x\n",InputEvent.Param1,InputEvent.Param2);
                            #endif
                            if ( (InputEvent.Param1 == 0x1000) &&
(InputEvent.Param2 ==       0x10))
                            {
                                    #ifdef NOISY
                                    printf("[SPREETA] You're in PDA Mode, install
your SIM Card, Fool!!\n");
                                    #endif
                            }
                            else if (InputEvent.Param1 > 0x10000)
                            {
                                    #ifdef NOISY
                                    printf("[SPREETA] Successful Send! Sent ID:
%d; Sending the next one :-)\n",InputEvent.Param1 & 0xFFFF);
                                    #endif
                                    GetTextCursor(&CurrPoint);
                                    PrintTextToScreen(CurrPoint.yCoord,
CurrPoint.xCoord, "Message sent successfully.");
                                    CurrPoint.yCoord += TEXT_LINE_SPACING;
                                    SetTextCursor(&CurrPoint);
                                    SpawnNotification();
                            }
                            else /* dunno what this is but we'd better try again
*/
                            {
                                    #ifdef NOISY
                                    printf("[SPREETA] Unknown Event,
retry...\n");
                                    #endif
                                    if (gNotificationStatus   ==
NOTIFICATION_IN_PROGRESS)
                                    {
                                            GetTextCursor(&CurrPoint);
                                            PrintTextToScreen(CurrPoint.yCoord,
CurrPoint.xCoord, "Message Failed; Retrying...");
                                            CurrPoint.yCoord += TEXT_LINE_SPACING;
                                            SetTextCursor(&CurrPoint);
                                            SpawnNotification();
                                    }
                            }
                    break;
            default:
                    #ifdef NOISY
                    printf("[SPREETA] SystemEvent %d; SubEvent
0x%x:\n",InputEvent.EventCode, InputEvent.Param1);
                    #endif
                    if (gNotificationStatus   == NOTIFICATION_IN_PROGRESS)
                            SpawnNotification();
                    break;
    }
}

/*************************************************************************
 * UpdateStatistics(algorithm_mode)
 *
 * This takes a long time with all the floating point arithmetic
 * so do the work when there's not any timing-sensitive conversions
 * taking place...
 *
 * Even so, don't take any more time in this routine than necessary,
 * to minimize time, only update one statistic per call to this routine.
 * Use gSampleCount to determine which statistic to update
 *
 * also save off the spreeta pixel values in case we switch to differential
 * mode during the next set of captures
 *
 ****************************************************************/
void UpdateStatistics(int algorithm_mode)
{
      long          TotalValue;
      float         AverageValue;
      int                  i;
      char        pScreenValue[20]; /* space for the data to print */
      int                  x_print_offset = 0;  /* used to offset for negative 
number printing */

      /* in low-power mode (aka. slow) we'll update all statistics every call 
*/
      switch (gSampleCount & 0x00000003 & gHighSpeedMode) /* Low_Power_Mode = 
0 */
      {
          /* High_Speed_Mode = 0xFFFFFFFF */
             case 0x00000000:  /* update MAX value */
                    /* find the Max value */
                    for (i=0; i<TOTAL_SPREETA_PIXELS; i++)
                    {
                          if (algorithm_mode == DRAW_DIFFERENTIAL)
                              CompareMaxValue(gCurrentSpreetaValue[i] - 
gBaselinedSpreetaValue[i],i);
                          else
                              CompareMaxValue(gCurrentSpreetaValue[i],i);
                    }

                    if (gSpreetaMaxValue < 0)
                        x_print_offset = (TEXT_WIDTH-1);

                    /* update the MAX value */
                    sprintf(pScreenValue,"%-
05.3f",gSpreetaMaxValue*(float)3/SPREETA_ADC_3V_VALUE);

      UpdateValueOnScreen(Y_MAX_LABEL+TEXT_LINE_SPACING,X_AVG_MIN_MAX_VALUE-
x_print_offset,(string)pScreenValue);

                    /* update the MAX angle */
                    sprintf(pScreenValue,"%-05.3f",RIU_MIN + 
gSpreetaMaxValueLocation*((float)RIU_MAX - RIU_MIN)/(TOTAL_SPREETA_PIXELS-1));

      UpdateValueOnScreen(Y_MAX_LABEL+TEXT_LINE_SPACING*2,X_AVG_MIN_MAX_VALUE
,(string)pScreenValue);

                    #ifdef NOISY
                    printf("Max Value Voltage: %-05.3f; 
",gSpreetaMaxValue*(float)3/SPREETA_ADC_3V_VALUE);
                    printf("at pixel %d with RIU value of: 
%05.3f\n",gSpreetaMaxValueLocation,gSpreetaMaxValueLocation*((float)RIU_MAX - 
RIU_MIN)/(TOTAL_SPREETA_PIXELS-1));
                    #endif

                    if (gHighSpeedMode == HIGH_SPEED_MODE)
                          break; /* break out and don't execute case 
0x00000001 in high-speed mode */
```

```
            case 0x00000001:  /* update MIN value */
                    /* find the Min value */
                    for (i=0; i<TOTAL_SPREETA_PIXELS; i++)
                    {
                            if (algorithm_mode == DRAW_DIFFERENTIAL)
                                    CompareMinValue(gCurrentSpreetaValue[i] -
gBaselinedSpreetaValue[i],i);
                            else
                                    CompareMinValue(gCurrentSpreetaValue[i],i);
                    }

                    if (gSpreetaMinValue < 0)
                         x_print_offset = (TEXT_WIDTH-1);

                    /* update the MIN value */
                    sprintf(pScreenValue,"%-
05.3f",gSpreetaMinValue*(float)3/SPREETA_ADC_3V_VALUE);

        UpdateValueOnScreen(Y_MIN_LABEL+TEXT_LINE_SPACING,X_AVG_MIN_MAX_VALUE-
x_print_offset,(string)pScreenValue);
                    /* update the MIN angle */
                    sprintf(pScreenValue,"%-05.3f",RIU_MIN +
gSpreetaMinValueLocation*((float)RIU_MAX - RIU_MIN)/(TOTAL_SPREETA_PIXELS-1));

        UpdateValueOnScreen(Y_MIN_LABEL+TEXT_LINE_SPACING*2,X_AVG_MIN_MAX_VALUE
,(string)pScreenValue);
                    #ifdef NOISY
                    printf("Min Value Voltage was: %-05.3f;
",gSpreetaMinValue*(float)3/SPREETA_ADC_3V_VALUE);
                    printf("at pixel %d with RIU value of: %-
05.3f\n",gSpreetaMinValueLocation,gSpreetaMinValueLocation* ((float)RIU_MAX -
RIU_MIN)/(TOTAL_SPREETA_PIXELS-1));
                    #endif

                    if (gHighSpeedMode == HIGH_SPEED_MODE)
                            break; /* break out and don't execute case
0x00000002 in high-speed mode */

            case 0x00000002: /* update average value */
                    TotalValue = 0;
                    for (i=0; i<TOTAL_SPREETA_PIXELS; i++)
                    {
                         /* upate the Average value */
                            TotalValue += gCurrentSpreetaValue[i];
                    }

                    AverageValue = (TotalValue /
(float)TOTAL_SPREETA_PIXELS)*(float)3/SPREETA_ADC_3V_VALUE;
                    sprintf(pScreenValue,"%-05.3f",AverageValue);

        UpdateValueOnScreen(Y_AVG_LABEL+TEXT_LINE_SPACING,X_AVG_MIN_MAX_VALUE,(
string)pScreenValue);
                    if (gHighSpeedMode == HIGH_SPEED_MODE)
                            break; /* break out and don't execute case
0x00000003 in high-speed mode */

            case 0x00000003:  /* Print the time to the screen */
                    {
                            UpdateTimeOnScreen();
                            /* check for the max value crossing the threshold */
                            if ((algorithm_mode == DRAW_DIFFERENTIAL) &&
(gNotificationStatus == NOTIFICATION_ENABLED))
                            {
                                    if ((
(gPreviousScreenValue[gSpreetaMaxValueLocation] <=
gHorizontalCursor1.PositionDifferential)
```

```
                                    &&
(gPreviousScreenValue[gSpreetaMaxValueLocation] <=
gHorizontalCursor2.PositionDifferential))
                                || (
(gPreviousScreenValue[gSpreetaMinValueLocation] >=
gHorizontalCursor1.PositionDifferential)
                                    &&
(gPreviousScreenValue[gSpreetaMinValueLocation] >=
gHorizontalCursor2.PositionDifferential)))
                                {
                                        gThresholdCrossedCount++;
                                        #ifdef NOISY
                                        printf("Crossed the Threshold! Count:
%d\n",gThresholdCrossedCount);
                                        #endif
                                        /* if we've been sitting outside the
threshold range long enough, tell somebody! */
                                        if (gThresholdCrossedCount >=
MAX_CONSECUTIVE_THRESHOLD_VIOLATIONS)
                                        {
                                                Point_t CurrPoint;
                                                char *PopUpMessage;

                                                SpawnNotification();
                                                /* dummy call to make sure we're
still processing input *and*
                                                 that we can escape out if
there's trouble... */
                                                GetTextCursor(&CurrPoint);
                                                CapturePopUpText(&PopUpMessage,
CurrPoint);
                                                /* I don't care why I came back,
the game's over */
                                                free(PopUpMessage);
                                                gNotificationStatus =
NOTIFICATION_DORMANT;
                                                DeleteAllSpreetaMailObjs();
                                                gThresholdCrossedCount = 0;
                                                display_differential_screen();
/* destroy pop-up by re-drawing the differential screen */
                                        }
                                }
                                else
                                        gThresholdCrossedCount = 0;
                        }
                }
                break;
        } /* end switch */

        /* update the sample count to provide a visual of how fast the samples
are running */
        sprintf(pScreenValue,"%d",gSampleCount++);
        UpdateValueOnScreen(Y_SAMPLE_COUNT_LABEL,X_SAMPLE_LABEL+X_SAMPLE_COUNT,
(string)pScreenValue);

}


/**************************************************************************
 * PixelToScreen()
 *
 * draws the pixel value to the screen
 *
 * works in two modes:
 *
 * a) in differential mode...
 * b) in aboslute mode...
 *
 * in either case, this route also maintains the "previous data" buffer
 * so it can erase the previous pixel data
 *
 ****************************************************************/
void PixelToScreen(int screen_pixel_value, int pixel_count, int
algorithm_mode)
{

       u_int16 CapturedScreenValue;  /* distinquish between Zoom & Abolute */

       #ifdef VERY_NOISY
       printf("Entering PixelToScreen with screen_pixel_value:
%d\n",screen_pixel_value);
       #endif

       if (algorithm_mode == DRAW_DIFFERENTIAL)
       {
           /* erase the previously drawn pixel */
              if (CheckEraseAtCursor(pixel_count,
gDifferentialCenterLine.PositionDifferential, gDifferentialCenterLine.Color)
!= ERASURE_COMPLETE)
                     if (CheckEraseAtCursor(pixel_count,
gHorizontalCursor1.PositionDifferential, gHorizontalCursor1.Color) !=
ERASURE_COMPLETE)
                            if (CheckEraseAtCursor(pixel_count,
gHorizontalCursor2.PositionDifferential, gHorizontalCursor2.Color) !=
ERASURE_COMPLETE)
                                   /* no interference detected, erase in
white... */

WriteLine(&DefaultRenderingContext,X_VERT_AXIS+1+pixel_count,gPreviousScreenVa
lue[pixel_count],

X_VERT_AXIS+1+pixel_count,gPreviousScreenValue[pixel_count]+1,COLOR_WHITE);

              if (   (screen_pixel_value !=
gDifferentialCenterLine.PositionDifferential)
                  && (screen_pixel_value !=
gHorizontalCursor1.PositionDifferential)
                  && (screen_pixel_value !=
gHorizontalCursor2.PositionDifferential)
                 )
              {
                  /* draw the new pixel */

WriteLine(&DefaultRenderingContext,X_VERT_AXIS+1+pixel_count,screen_pixel_valu
e,

X_VERT_AXIS+1+pixel_count,screen_pixel_value+1,COLOR_BLACK);
               } /* end test for no interference */
       }
       else
       {
              if (gSpreetaPixelsBaselined == TRUE)
              {
                     if (algorithm_mode == DRAW_ABSOLUTE)

       CapturedScreenValue=gCapturedAbosluteScreenValue[pixel_count];
                     else

       CapturedScreenValue=gCapturedZoomScreenValue[pixel_count];
              }
              else /* no capture of the data yet, don't use the data in the
array */
              {
                    CapturedScreenValue = screen_pixel_value; /* use the
current value until we get a capture */
             }

          /* erase the previously drawn pixel */
             if (CheckEraseAtCursor(pixel_count,
gHorizontalCursor1.PositionAbsolute, gHorizontalCursor1.Color) !=
ERASURE_COMPLETE)
                    if (CheckEraseAtCursor(pixel_count,
gHorizontalCursor2.PositionAbsolute, gHorizontalCursor2.Color) !=
ERASURE_COMPLETE)

WriteLine(&DefaultRenderingContext,X_VERT_AXIS+1+pixel_count,gPreviousScreenVa
lue[pixel_count],

X_VERT_AXIS+1+pixel_count,gPreviousScreenValue[pixel_count]+1,COLOR_WHITE);

             /* draw the originally captured pixel in the graph, Black */
             if (   (CapturedScreenValue !=
gHorizontalCursor1.PositionAbsolute)
                 && (CapturedScreenValue !=
gHorizontalCursor2.PositionAbsolute)
                )

WriteLine(&DefaultRenderingContext,X_VERT_AXIS+1+pixel_count,CapturedScreenVal
ue,

X_VERT_AXIS+1+pixel_count,CapturedScreenValue+1,COLOR_BLACK);

             /* draw the new value of the pixel in the graph, Red */
             if (CapturedScreenValue != screen_pixel_value) /* is new
different from old? */
                    if (   (screen_pixel_value !=
gHorizontalCursor1.PositionAbsolute)
                        && (screen_pixel_value !=
gHorizontalCursor2.PositionAbsolute)
                       )

WriteLine(&DefaultRenderingContext,X_VERT_AXIS+1+pixel_count,screen_pixel_valu
e,

X_VERT_AXIS+1+pixel_count,screen_pixel_value+1,COLOR_RED);
       }
       /* update the previous value array */
       gPreviousScreenValue[pixel_count] = screen_pixel_value;

}

/**************************************************************************
 * signal intercept routine
 *
 * upon receiving an asyncronous signal, this routine will handle the signal
 * and take the appropriate action.  Note that we don't expect any input
 * on this channel, but better safe than sorry.
 *
 * For debug, we can always type ^C and if the signal handler is running,
 * we can return control to the OS, run cleanup, etc....
 *
 ****************************************************************/
void sighandler (int signal)
{

    _os_sigmask (1);                /* mask signals during signal handler */

       #ifdef NOISY
       printf("in sighandler with signal 0x%08x\n",signal);
       #endif
    switch (signal)
    {
        case SIGINT:                    /* ^C = keyboard interrupt signal */
             case SIGQUIT:                    /* ^E = keyboard abort signal */
                    #ifdef NOISY
            printf("Termination signal received\n");
                    #endif
                    deinitalize_system();
            exit (signal);
            break;

        case SIG_QUEUE:
                    #ifdef NOISY
            printf("SIG_QUEUE signal received\n");
                    #endif
            break;

        default:
                    #ifdef NOISY
            printf("Unknown signal received ==> %d\n", signal);
                    #endif
                    break;
    }

    _os_sigmask (-1);       /* unmask signals upon exit of signal handler */
    _os_rte();              /* cannot use 'return', must use _os_rte() upon
                             * return from signal handler */
}

/********************************************************************
 *
 * tmode_nopause()
 *
 * this function turns off screen pause mode so the scrolling of the debug
outputs
 * won't get stuck due to lack of input from stdin
 *
 * equivalent to 'tmode nopause' from shell
 *
 ********************************************************************/
void tmode_nopause(void)
{
    struct sgbuf termopts;

    _gs_opt(1, &termopts);
    termopts.sg_pause = 0;
    _ss_opt(1, &termopts);
}

/************************************************************
 *
 * delay()
 *
 * pass in the value of "duration" which is computed in terms
 * of "ticks"
 *
 * a tick is 10mSec in this instantiation of the OS.
 *
 * special cases: tick = 0: sleep forever
 *                tick = 1; give up the current time slice
 *
 ************************************************************/
void delay(int duration)
{
      u_int32 sleepval;
    sleepval = duration;
       while (sleepval > 1)   /* if we exit sleep early, re-invoke until
duration elapses */
       {
              _os9_sleep(&sleepval);
              #ifdef NOISY
              printf("exited _os9_sleep() with a value of %d\n",sleepval);
              #endif
       }
}
/***************************************************************'
 *
 * CapturePopUpText()
 *
 * gather the input from the user
 *
 * Returns:
 *   SUCCESS - if the user does't cancel out
 *   FAILURE - if the user does cancel out
 *
 * Captured Inputs:
 * 1) standard keys for printing and adding to the string
 * 2) TAB - end of input
 * 3) Enter - make a new line
 * 4) Escape - cancel out of the function and return "FAILURE"
 * 5) BackSpace - back up one character
 *
 *****************************************************************/
int CapturePopUpText(char **CaptureString, Point_t CurrPoint)
{
       char      *TextLines[MAX_POPUP_TEXT_LINES];  /* lines of text in the
popup */
       u_char    input_data;       /* the character input from the keypad */
    eventmsg  RcvMsg;           /* the input Queue for the keypad */
       int       CurLine = 0;      /* the current line being entered by the
user */
       int            ContinueInput;
       int            ReturnValue = ERROR_NO_TEXT_CAPTURE;

       #ifdef NOISY
       printf("Enter CapturePopUpText; CurrPoint.x:%d;
CurrPoint.y:%d\n",CurrPoint.xCoord,CurrPoint.yCoord);
       #endif

       TextLines[CurLine] = malloc(MAX_POPUP_LINE_LENGTH);
       *TextLines[CurLine] = 0x00;       /* null out the string */
       ContinueInput = TRUE;

    while ((ContinueInput == TRUE) && (ReadMsg(&RcvMsg) == SUCCESS)) /*
blocking read */
       {
              if (    (RcvMsg.EventCode != EVENT_KEY)
                   && (RcvMsg.EventCode != EVENT_SYSKEY)
                   && (RcvMsg.EventCode != EVENT_MODIFIER_STATE_CHANGE))
              {
                     #ifdef NOISY
                     printf("Received System event %d
(0x%x)\n",RcvMsg.EventCode,RcvMsg.EventCode);
                     #endif
                     ProcessSystemEvent(RcvMsg);
              }
              else
              {
                     input_data = (u_char)RcvMsg.Param1;
                     #ifdef NOISY
                     printf("Pop-Up Capture of \"%c\"
(0x%02x)\n",input_data,input_data);
```

```
                    #endif

                    switch (input_data)
                    {
                          case TAB_KEY:
                                ContinueInput = FALSE; /* User has ended the
input */
                                {
                                      int i;
                                      *CaptureString = malloc((CurLine+1)*
MAX_POPUP_LINE_LENGTH);
                                      **CaptureString = 0x00; /* NULL
Terminate the string */
                                      for (i=0; i<= CurLine; i++)      /*
return the user's data */
                                      {
                                            strcat(*CaptureString,
TextLines[i]);
                                            #ifdef NOISY
                                            printf("Line[%d]:
(%s)\n",i,TextLines[i]);
                                            #endif
                                            free(TextLines[i]);
                                      }
                                      #ifdef NOISY
                                      printf("CaptureString:
(%s)\n",*CaptureString);
                                      #endif
                                }
                                ReturnValue = SUCCESS;
                                break;
                          case RETURN_KEY:
                                /* terminate current line and generate a new
one */
                                strncat(TextLines[CurLine],(const char
*)&input_data,1);
                                CurLine++;
                                TextLines[CurLine] =
malloc(MAX_POPUP_LINE_LENGTH);
                                *TextLines[CurLine] = 0x00;
                                /* position the cursor at the beginning of
the new line */
                                EraseTextCursor();
                                CurrPoint.xCoord =
POPUP_TOP_X+POPUP_BOARDER_THICKNESS+CURSOR_WIDTH;
                                CurrPoint.yCoord += TEXT_LINE_SPACING;
                                SetTextCursor(&CurrPoint);
                                PrintTextCursor();
                                #ifdef NOISY
                                printf("Creating Line No: %d\n",CurLine);
                                #endif
                                break;
                          case BACKSPACE_KEY:
                                #ifdef NOISY
                                printf("Erasing Last Character\n");
                                #endif
                                EraseLastCharacter(CurrPoint,
(string)TextLines[CurLine]);
                                break;
                          case EXIT_KEY:
                          case HOME_KEY:
                          case 0x84:
                          case 0x8C:
                          case 0x8D: /* press & hold Home key */
                          case 0x8F: /* sometimes it's an 0x8F, dunno why?? */
                                #ifdef NOISY
                                printf("Escape From User Input\n");
                                #endif
                                ContinueInput = FALSE; /* User has ended the
input */
                                ReturnValue = ERROR_NO_TEXT_CAPTURE;
                                break;
                          default:
                                if (  (input_data >= 0x20)   /* printable key
*/
                                   && (input_data <= 0x80))
                                {
                                      Point_t CheckPoint;

                                      strncat(TextLines[CurLine],(const char
*)&input_data,1);

                                      /* wrap to the next row if at the edge
of the pop-up */
                                      GetTextCursor(&CheckPoint);
                                      if (CheckPoint.xCoord > POPUP_BOTTOM_X
- POPUP_BOARDER_THICKNESS - TEXT_WIDTH*2)
                                      {
                                            EraseTextCursor();

      PrintTextToScreen(CurrPoint.yCoord, CurrPoint.xCoord,
(string)TextLines[CurLine]);
                                            CurLine++;
                                            TextLines[CurLine] =
malloc(MAX_POPUP_LINE_LENGTH);
                                            *TextLines[CurLine] = 0x00;
                                            /* position the cursor at the
beginning of the new line */
                                            CurrPoint.xCoord =
POPUP_TOP_X+POPUP_BOARDER_THICKNESS+CURSOR_WIDTH;
                                            CurrPoint.yCoord +=
TEXT_LINE_SPACING;
                                            SetTextCursor(&CurrPoint);
                                            PrintTextCursor();
                                            #ifdef NOISY
                                            printf("Wrapping to the next
line: %d\n",CurLine);
                                            #endif
                                      }
                                      else /* not at end of line; no need to
wrap */
                                      {
                                            PrintNextCharacter( CurrPoint,
(string)TextLines[CurLine]);
                                      }
                                }
                                else
                                {
                                      #ifdef NOISY
                                      printf("Sorry, that particular key
doesn't do anything...\n");
                                      #endif
                                }
                                break;
                    } /* end case */
              } /* end else */
      } /* end if */
      return(ReturnValue);
} /* end CapturePopUpText() */

/***********************************************************'
 *
 * ProcessInput()
 *
 * used to determine changes to the program while running.
 *
 * in the simplest form. we have four basic commands:
 *
 * s - start reading spreeta data
 * a - display the data in "absolute" mode
 * d - display the data in "differential" mode
 * c - capture the Spreeta Data  (prerequisite for Differential Mode)
 * p - print the current screen to memory
 * j - move graph down by decreasing integration period
 * k - move graph up by increasing integration period
 * l - change to low-power mode (e.g. sample less often)
 * h - change to high speed mode (e.g. smaple more often)
 * n - enable wireless notification
 * q - quit the program
 *
 *******************************************************************/
int ProcessUserInput( u_char input_data, int *algorithm_mode )
{
      int       return_val,i;

      #ifdef NOISY
      printf("crazy user pressed a \"%c\" (0x%02x)\n",input_data,input_data);
      fflush(stdout);
      #endif

      switch (input_data)
      {
            case 's':
            case 'S':
            case 'a':
            case 'A':
                  *algorithm_mode = DRAW_ABSOLUTE;
                  return_val = RESTART_ALGORITHM;
                  gScreenMaxValue = SPREETA_ADC_3V_VALUE;
                  gScreenMinValue = 0;
                  break;
            case 'd':
            case 'D':
                  if (gSpreetaPixelsBaselined == FALSE) /* run a capture
now... */
                  {
                        for (i=0; i<TOTAL_SPREETA_PIXELS; i++)
                        {
                              /* save off the data values for
differential mode calculations */
                              gBaselinedSpreetaValue[i] =
SmoothingAlgorithm(i);
                        }
                  }
                  gSpreetaPixelsBaselined = TRUE;
                  *algorithm_mode = DRAW_DIFFERENTIAL;
                  return_val = RESTART_ALGORITHM;
                      /* screen Max & Min have no meaning in Differential
Mode */
                  gScreenMaxValue = SPREETA_ADC_3V_VALUE;
                  gScreenMinValue = 0;
                  break;
            case '0': case '1': case '2': case '3': case '4':
            case '5': case '6': case '7': case '8': case '9':
                  if (*algorithm_mode == DRAW_DIFFERENTIAL) /* numbers
adjust differential zoom only */
                  {
      /* while differential screen being displayed */
                        return_val = RESTART_ALGORITHM;   /* force a re-
draw of the screen */
```

```
                            gDifferentialTicksPerPixelPower =  input_data -
'0';
                        }
                        else
                        {
                                return_val = CONTINUE_ALGORITHM;
                        }
                        break;
            case 'z':
            case 'Z':
                        *algorithm_mode = DRAW_ZOOM;
                        return_val = RESTART_ALGORITHM;
                            /* calculate screen Max & Min limits */
                              gScreenMaxValue = gSpreetaMaxValue +
SCREEN_MAX_OFFSET;
                              gScreenMinValue = gSpreetaMinValue -
SCREEN_MIN_OFFSET;
                        /* need to adjust cursor position values here
                           and adjust 'em back when changing to aboslute screen
*/
                        break;
            case 'c': /* Caputre */
            case 'C':
                        /* don't change the algorithm mode */
                        return_val = CONTINUE_ALGORITHM;;
                        for (i=0; i<TOTAL_SPREETA_PIXELS; i++)
                        {
                                /* save off the data values for differential mode
calculations */
                                gBaselinedSpreetaValue[i] =
SmoothingAlgorithm(i);
                          gCapturedAbosluteScreenValue[i] =
ProcessPixelValue(i,DRAW_ABSOLUTE);
                          gCapturedZoomScreenValue[i] =
ProcessPixelValue(i,DRAW_ZOOM);
                        }
                        gSpreetaPixelsBaselined = TRUE;
                        gSampleCount = 0; /* start the sample count over at 0 */

UpdateValueOnScreen(Y_SAMPLE_COUNT_LABEL,X_SAMPLE_LABEL+X_SAMPLE_COUNT,"0
");
                        break;
            case 'j':
            case 'J': /* move waveform down by decreasing integration period
*/
                        /* don't change the algorithm mode */
                        return_val = CONTINUE_ALGORITHM;
                        gIntegrationPeriod -= INTEGRATION_INCREMENT;
                        break;
            case 'k':
            case 'K': /* move waveform up by increasing integration period */
                        /* don't change the algorithm mode */
                        return_val = CONTINUE_ALGORITHM;
                        gIntegrationPeriod += INTEGRATION_INCREMENT;
                        break;
            case 'l':
            case 'L': /* change to low-power mode by enabling sleep */
                        /* don't change the algorithm mode */
                        return_val = CONTINUE_ALGORITHM;
                        gHighSpeedMode = LOW_POWER_MODE;
                        break;
            case 'h':
            case 'H': /* change to High-Speed mode by disabling sleep */
                        /* don't change the algorithm mode */
                        return_val = CONTINUE_ALGORITHM;
                        gHighSpeedMode = HIGH_SPEED_MODE;
                        break;
```

```
             case 'n':
             case 'N': /* enable wireless notification */
                    {
                          char *eMailAddressList;
                          char *NotificationMessage;
                          Point_t CurrPoint;

                           /* get user's input for address and message */
                           CurrPoint = DrawPopUp("Enter E-Mail Address(es)");
                           if (CapturePopUpText(&eMailAddressList, CurrPoint)
== SUCCESS)
                           {
                                 CurrPoint = DrawPopUp("Enter Notificaiton
Message");
                                 if
(CapturePopUpText(&NotificationMessage,CurrPoint) == SUCCESS)
                                 {
                                        struct SMSNotification
*pLocalSMSNotification; /* pointer to linked list of SMS messages */
                                        char *eMailAddress, *ListChar,
*FillChar;

                                        DeleteAllSpreetaMailObjs();  /* clean
out all lingering orphans */
                                        gNotificationStatus = TRUE; /* set
notification to TRUE */
                                        gpSMSNotificationList = NULL;
                                        eMailAddress =
malloc(MAX_POPUP_LINE_LENGTH);
                                        *eMailAddress = 0x00;
                                        FillChar = eMailAddress;
                                        ListChar = eMailAddressList;
                                        do
                                        {
                                               while((*ListChar != '\n') &&
(*ListChar != 0x00))
                                               {
                                                      *FillChar++ =
*ListChar++; /* copy the next e-mail address from the list */
                                               }
                                               *FillChar = 0x00; /* terminate
the string */
                                               #ifdef NOISY
                                               printf("E-Mail Address:
(%s)\n",eMailAddress);
                                               #endif

                                               if (*ListChar != 0x00)
                                                    ListChar++;  /* skip the
newline */

                                               if (strchr(eMailAddress,'@')
!= NULL)     /* only process valid e-mail addresses */
                                               {
                                                      pLocalSMSNotification =
malloc(sizeof(struct SMSNotification));
                                                      pLocalSMSNotification-
>SMS_ID = BuildSMSMessage(eMailAddress, NotificationMessage);
                                                      pLocalSMSNotification-
>Sent = FALSE;
                                                      pLocalSMSNotification-
>Address = malloc(strlen(eMailAddress));

strcpy(pLocalSMSNotification->Address,eMailAddress);
                                                      pLocalSMSNotification-
>Next = gpSMSNotificationList;
```

```
                                                        gpSMSNotificationList =
pLocalSMSNotification;
                                                        FillChar = eMailAddress;
/* set up for the next address copy */
                                                   }
                                              }
                                              while(*ListChar != 0x00);
                                            #ifdef NOISY
                                              printf("Finished building SMS Message
List: (");
                                              for ( pLocalSMSNotification =
gpSMSNotificationList;
                                                    pLocalSMSNotification != NULL;
                                                    pLocalSMSNotification =
pLocalSMSNotification->Next)
                                                     {

printf("%d,",pLocalSMSNotification->SMS_ID);
                                                     }
                                              printf(")\n");
                                            #endif
                                              free(NotificationMessage);
                                         }
                                         free(eMailAddressList);
                                    }
                      }

                      /* erase pop-up by re-drawing the screen */
                      return_val = RESTART_ALGORITHM;

                      break;
             case DOWN_KEY:  /* move the active cursor down one pixel */
             case 0xC2:
                      /* don't change the algorithm mode */
                      return_val = CONTINUE_ALGORITHM;

EraseHorizontalCursor(gHorizontalCursorActive,*algorithm_mode);
                      if (*algorithm_mode == DRAW_DIFFERENTIAL)
                      {
                                 gHorizontalCursorActive-
>PositionDifferential++;
                                 if ((gHorizontalCursorActive-
>PositionDifferential + gHorizontalCursorActive->Width)
                                      ==
(gDifferentialCenterLine.PositionDifferential - 1))
                                     /* the cursor has invaded the space of
the differential line so jump over it... */
                                       gHorizontalCursorActive-
>PositionDifferential = gDifferentialCenterLine.PositionDifferential +
gDifferentialCenterLine.Width + 2;

                                 /* don't let it run off the bottom of the
screen */
                                 if (gHorizontalCursorActive-
>PositionDifferential >= (BOTTOM_GRAPH_BOUNDARY-(gHorizontalCursorActive-
>Width-1)))
                                       gHorizontalCursorActive-
>PositionDifferential--;
                      }
                      else
                      {
                            gHorizontalCursorActive->PositionAbsolute++;

                            /* don't let it run off the bottom of the screen
*/
                            if (gHorizontalCursorActive->PositionAbsolute >=
(BOTTOM_GRAPH_BOUNDARY-(gHorizontalCursorActive->Width-1)))
```

```
                                gHorizontalCursorActive->PositionAbsolute--;
                      }

DrawHorizontalCursor(gHorizontalCursorActive,*algorithm_mode);
                      break;
             case UP_KEY:  /* move the active cursor up one pixel */
             case 0xC3:

EraseHorizontalCursor(gHorizontalCursorActive,*algorithm_mode);
                      /* don't change the algorithm mode */
                      return_val = CONTINUE_ALGORITHM;
                      if (*algorithm_mode == DRAW_DIFFERENTIAL)
                      {
                            gHorizontalCursorActive->PositionDifferential--;
                            if ((gHorizontalCursorActive->PositionDifferential
- 1)
                                 ==
(gDifferentialCenterLine.PositionDifferential +
gDifferentialCenterLine.Width))
                                /* the cursor has invaded the space of the
differential center line so jump over it... */
                                  gHorizontalCursorActive-
>PositionDifferential = gDifferentialCenterLine.PositionDifferential -
gHorizontalCursorActive->Width - 2;

                            /* don't let it run off the top of the screen */
                            if (gHorizontalCursorActive->PositionDifferential
<= TOP_GRAPH_BOUNDARY)
                                  gHorizontalCursorActive-
>PositionDifferential++;
                      }
                      else
                      {
                            gHorizontalCursorActive->PositionAbsolute--;
                            /* don't let it run off the top of the screen */
                            if (gHorizontalCursorActive->PositionAbsolute <=
TOP_GRAPH_BOUNDARY)
                                gHorizontalCursorActive->PositionAbsolute++;
                      }

DrawHorizontalCursor(gHorizontalCursorActive,*algorithm_mode);
                      break;
             case TAB_KEY:  /* Switch the active cursor */
                      /* don't change the algorithm mode */
                      return_val = CONTINUE_ALGORITHM;
                      /* deselect current cursor */
                      gHorizontalCursorActive->Color = CURSOR_DEFAULT_COLOR;
                      /* switch active cursor */
                      gHorizontalCursorActive = gHorizontalCursorActive->Next;
                      gHorizontalCursorActive->Color = CURSOR_SELECTED_COLOR;
                      /* re-draw the cursors: no need to erase first because
they're not moving */

DrawHorizontalCursor(&gHorizontalCursor1,*algorithm_mode);

DrawHorizontalCursor(&gHorizontalCursor2,*algorithm_mode);
                      break;
             case 'p':
             case 'P':
                      /* print the current screen to non-volatile storage */
                      return_val = CONTINUE_ALGORITHM;
                      {
                            Bitmap_t Bitmap;
                            u_int8   DummyReadBuff[8]; /* read buffer to
search for unused bitmap */
                            fisID_t  FISid;
                            char     BitmapFileName[20];
```

```
                              u_int32  BuffLen;

                              #ifdef NOISY
                                  u_int32  minalloc,  /* min. allocation size */
                                           memsegs,          /* number of memory
segments */
                                           totramst,  /* total ram at startup */
                                           ctotram;          /* current total
ram */

                                     _os_get_blkmap(NULL, NULL, 0,
&minalloc,&memsegs, &totramst, &ctotram);
                              printf("Total Free RAM upon entry to PrintScreen:
%d\n",ctotram);
                              #endif /* NOISY */

                              /* capture the current screen */
                           ScreenToBitmap(&DefaultRenderingContext, NULL,
&Bitmap, UN_COMPRESSED_ROB_FILE_TYPE);
                           /* ScreenToBitmap(&DefaultRenderingContext, NULL,
&Bitmap, ROB_FILE_COMPRESSION_METHOD_RC2);    */

                              FISid = BITMAP_FIS_ID;
                              BuffLen = sizeof(DummyReadBuff);

                              /* search for a free ID to store the bitmap */
                              while (FISRead(BITMAP_FIS_TYPE, FISid,
DummyReadBuff,&BuffLen) != E_ITEM_NOT_FOUND)
                                  FISid++;


sprintf(BitmapFileName,"FIS:%d.%d",BITMAP_FIS_TYPE,FISid);
                              StoreBitmap( (string)BitmapFileName, &Bitmap );
                              /* DestroyBitmap(&Bitmap); */ /* free up the
memory for the compressed version */

                              #ifdef NOISY
                                     _os_get_blkmap(NULL, NULL, 0,
&minalloc,&memsegs, &totramst, &ctotram);
                              printf("Total Free RAM upon exit from PrintScreen:
%d\n",ctotram);
                              #endif /* NOISY */
                       }
                       break;
              case 'q':
              case 'Q':
                       *algorithm_mode = -1;  /* doesn't really matter, but
send SOMETHING */
                       return_val = QUIT_ALGORITHM;
                       break;
              case 't':
              case 'T': /* threshold violation invoked by user */
                       return_val = RESTART_ALGORITHM;
                       if (gNotificationStatus != NOTIFICATION_DORMANT)
                       {
                       int ContinueInput = TRUE;
                              eventmsg RcvMsg;
                              char *PopUpMessage;
                              Point_t CurrPoint;

                       CopyBitmapToScreen(SIMULATION_ABSOLUTE_FILE);
                              UpdateDateOnScreen();
                              UpdateTimeOnScreen();
                           while ((ContinueInput == TRUE) && (ReadMsg(&RcvMsg)
== SUCCESS)) /* blocking read */
                              {
                                     if (    (RcvMsg.EventCode == EVENT_KEY)
                                        || (RcvMsg.EventCode == EVENT_SYSKEY)
                                        || (RcvMsg.EventCode ==
EVENT_MODIFIER_STATE_CHANGE))
                                    {
                                          ContinueInput = FALSE;
                                    }
                              }
                      CopyBitmapToScreen(SIMULATION_DIFFERENTIAL_FILE);
                              UpdateDateOnScreen();
                              UpdateTimeOnScreen();
                              ContinueInput = TRUE;
                           while ((ContinueInput == TRUE) && (ReadMsg(&RcvMsg)
== SUCCESS)) /* blocking read */
                              {
                                    if (    (RcvMsg.EventCode == EVENT_KEY)
                                        || (RcvMsg.EventCode == EVENT_SYSKEY)
                                        || (RcvMsg.EventCode ==
EVENT_MODIFIER_STATE_CHANGE))
                                    {
                                          ContinueInput = FALSE;
                                    }
                              }
                              SpawnNotification();

                              /* dummy call to make sure we're still processing
input *and*
                                 that we can escape out if there's trouble... */
                              GetTextCursor(&CurrPoint);
                              CapturePopUpText(&PopUpMessage, CurrPoint);
                           /* I don't care why I came back, the game's over */
                              free(PopUpMessage);
                              gNotificationStatus = NOTIFICATION_DORMANT;
                              DeleteAllSpreetaMailObjs();

                      }
                      break;
              default:
                      return_val = CONTINUE_ALGORITHM;
                      #ifdef NOISY
                      printf("Sorry, that particular key doesn't do
anything...\n");
                      #endif
                      break;

         } /* end case */

    if (return_val == RESTART_ALGORITHM)
       {
           /* re-initalize the globals */
           #ifdef NOISY
           printf("gScreenMaxValue = %d; gScreenMinValue =
%d\n",gScreenMaxValue,gScreenMinValue);
           printf("gScreenMaxValue-gScreenMinValue =
%f\n",(float)(gScreenMaxValue-gScreenMinValue));
           #endif NOISY
           gConvertADCToScreen = (Y_BOT_VERT_LABEL-
Y_TOP_VERT_LABEL)/(float)(gScreenMaxValue-gScreenMinValue);

           if (gScreenMinValue < 0) /* floor funciton for minimum screen value
*/
               gScreenMinValue=0;

           if (gScreenMaxValue > SPREETA_ADC_3V_VALUE)      /* celing funciton
for maximum screen value */
               gScreenMaxValue=SPREETA_ADC_3V_VALUE;
    }
    return(return_val);
```

```
}

/*********************************************************************
 *
 * Main()
 *
 * the main algorithm is:
 * 1) initalize system
 * 2) read pixels from Spreeta
 * 3) draw pixels to screen
 *
 **********************************************************************/
int main(int argc, string argv[])
{
      int     pixel_count;         /* current pixel number for read from
spreeta */
      int     algorithm_status;  /* tells when to re-start the inner loop or
quit the program */
      int     algorithm_mode;           /* tells absolute mode vs.
differential mode */
      int     quit = FALSE;        /* loop controlling variable */
      int          PixelValue;
    eventmsg  RcvMsg;         /* events from keypad, MDS, etc. */

      algorithm_status = initalize_system(&algorithm_mode);            /* does
all init for keypad, LCD and Spreeta device */

    while (quit != TRUE)
      {
             MASK_INTERRUPTS;  /* keep the timing for the read as consistant
as possible */
             START_COUNTER;  /* for exact sample control, use a dedicated,
very precise timer */
        SampleSpreeta(BEGIN_INTEGRATION); /* start an integration period */
             while (*TMR2_COUNTER < gIntegrationPeriod)
             {
            /* make sure this loop isn't optimized out by the compiler:
               the "volatile" declaration of TMR2_COUNTER should be sufficient
*/
                    #ifdef NOISY_INTS_NOT_MASKED
                    printf("%d<%d\n",*TMR2_COUNTER,gIntegrationPeriod);
                    #endif
             }

      SampleSpreeta(CAPTURE_PIXELS);    /* sample now that integration period
has expried */
             RESTORE_INTERRUPTS; /* re-enable interrupts */
             RESET_COUNTER; /* turn off the counter to save power and restart
it at zero */

             /* draw the samples to the screen */
             for (pixel_count = 0;pixel_count < TOTAL_SPREETA_PIXELS;
pixel_count++)
             {

                    PixelValue =
ProcessPixelValue(pixel_count,algorithm_mode);

                    PixelToScreen(PixelValue,pixel_count,algorithm_mode);
             }

             UpdateStatistics(algorithm_mode);

             if (gHighSpeedMode == LOW_POWER_MODE)
             {
                    #ifdef NOISY
                    printf("Entering sleep to save power\n");
                    #endif
                    delay(SLEEP_HALF_SECOND);
             }

             /* check for new messages: Key events and MDS responses */
          if (GetMsg(&RcvMsg) == SUCCESS)
             {
                    switch (RcvMsg.EventCode)
                    {
                           case EVENT_KEY:
                           case EVENT_SYSKEY:
                           case EVENT_MODIFIER_STATE_CHANGE:
                                  #ifdef NOISY
                                  printf("Received Keypad Input event %d
(0x%x)\n",RcvMsg.EventCode,RcvMsg.EventCode);
                                  #endif
                           algorithm_status =
ProcessUserInput((u_char)RcvMsg.Param1, &algorithm_mode);  /* non-blocking
call to see if the user wants anything... */
                                  /* process algorithm status changes from user
input (if any) */
                           switch (algorithm_status)
                           {
                                         case CONTINUE_ALGORITHM:
                                                break;

                                         case RESTART_ALGORITHM:
                                                  switch(algorithm_mode)
                                                  {
                                                           case DRAW_ABSOLUTE:

       display_absolute_screen();
                                                                    break;
                                                           case DRAW_ZOOM:

       display_zoom_screen();
                                                                    break;
                                                           case
DRAW_DIFFERENTIAL:

       display_differential_screen();
                                                                    break;
                                                }
                                                break;
                                         case QUIT_ALGORITHM:
                                                  quit = TRUE;    /* end the
outer loop */
                                                break;
                                  } /* end switch */
                                  break;
                           default:
                                  ProcessSystemEvent(RcvMsg);
                                  break;
                    }
             }
       } /* end outer While loop */

    deinitalize_system();
       reset();  /* let the system return to full Messaging mode */
    return 0; /* only here to squelch compiler warnings */
} /* main */
```

REFERENCES

[1] *Texas Instrument's Spreeta SPR Biosensor Technology Overview*, Texas Instruments Incorporated, Dallas, TX, 2003 [Online]. Available: http://www.ti.com/spreeta

[2] *Motorola's Accompli 009 Personal Communicator Product Overview*, 1994-2003 Motorola Inc. Corporate Headquarters Schaumburg, IL, 2003 [Online]. Available: http://www.motorola.com/accompli009

[3] Stubbs, D. D., Hunt, W.D., Lee, S.H. and Doyle, D.F. (2002). "Gas phase activity of anti-FITC antibodies immobilized on a surface acoustic wave resonator" in press, *Biosensors and Bioelectronics*.

[4] Sommers, D. R., Stubbs, D. D., Hunt, W.D., "*Investigation of SPR Technology Using Texas Instruments' SpreetaTM Sensor*", IEEE Sensors Conference, 2002

[5] *NDS-2000 Hand-held Drug Detector from Scintrex Trace Corp.*, Scintrex Trace Corp., wholly owned subsidiary of Control Screening LP, Fairfield, NJ, 2003 [Online]. Available: http://tracedetection.com/narcotics_detector.html

[6] Sigma-Aldrich Product Home page, "*Monoclonal Anti-FITC antibody produced in mouse***"** 2003 Sigma-Aldrich Inc. [Online]. Available: http://www.sigmaaldrich.com

[7] *US Customs & Border Protection*, US Department of Homeland Security, NW Washington, D.C., 2003 [Online]: Available: http://www.customs.ustreas.gov

[8] National Research Council: Division on Engineering and Physical Sciences, *Making the Nation Safer: The Role of Science and Technology in Countering Terrorism*, National Academy Press, 2002, pp. 113-117 Available: http://www.nap.edu/catalog/10415.html?onpi_newsdo062402

[9] S. P. Layne, T. J. Beugelsdijk, and C. K. N. Patel, *Firepower in the Lab: Automation in the Fight Against Infectious Diseases and Bioterrorism*, Joseph Henry Press, 2001, pp. 143-164 Available: http://books.nap.edu/books/0309068495/html/150.html#pagetop

[10] K. G. Ong, K. Zeng, C. A. Grimes, "A Wireless, Passive Carbon Nanotube-Based Gas Sensor", *IEEE Sensors Journal*, vol. 2, No. 2, April 2002 Available: http://www.ee.psu.edu/grimes/publications/IEEE_nanotube.pdf

[11] Kress-Rogers, E., *Biosensors and Electronic Noses* Chapter 7, CRC Press Inc., 1997

[12] *Nomadics' Surface Plasmon Resonance Evaluation Kit*, Nomadics Inc., Stillwater, OK , 2002 [Online]. Available: http://www.aigproducts.com/surface_plasmon_resonance/spr.htm

[13] Sigma-Aldrich Chemical Company, St. Louis, MO, 2002 [Online] Available: http://www.sigmaaldrich.com/

[14] NIST certification for Motorola Accompli 009 for FIPS 140-1 Available: http://cs-www.ncsl.nist.gov/cryptval/140-1/1401val.htm

[15] *Maxim Samples and Literature Request Page*, Maxim Integrated Products, Sunnyvale, CA, 2002 [Online]. Available: http://www.maxim-ic.com/samples

[16] *Texas Instruments Sample Request Page,* Texas Instruments Incorporated, Dallas, TX, 2002 [Online]. Available: http://www.ti.com/sc/docs/sampreq.htm

[17] TechTarget Network, *Hardware, Electronics, Dielectric Material,* TechTarget, Needham, MA 02494 [Online]. Available: http://whatis.techtarget.com/definition/0,,sid9_gci211945,00.html

[18] Validated FIPS 140-1 and FIPS 140-2 Cryptographic Modules, *140-1 and 140-2 Validation List; Certification number 260*, NIST Certification Website, 2002 [Online]. Available: http://csrc.nist.gov/cryptval/140-1/1401val2002.htm

[19] Messaging Sofware Plaforms, *Motorola Messaging Platform (MMP)*, 1994-2003 Motorola, Inc., Corporate Headquarters Schaumburg, IL, 2003 [Online]. Available: http://www.motorola.com/MSP/mmp/index.html

[20] Molecular Biology CyberLab, *Gel Electrophoresis of DNA*, University of Illinois, 2003, Urbana-Champaign [Online] Available: http://www.life.uiuc.edu/molbio/geldigest/electro.html