Chapter 6

Performance and Limitations

In this chapter we discuss the timing performance and the limitations of the real-time wavelength modulated optical fiber sensor systems.

6.1 Real-Time Performance of the Wavelength Modulated EFPI Sensor System

After loading all the real-time signal processing firmware and software, as listed in Appendix A.1, on the zero wait-state SRAM memory of the wavelength modulated EFPI sensor system, having the 40 MHz TMS320C40 DSP processor, we measured the execution times for all the signal processing functions. Given below are the function wise execution time of the real-time system.

Function Name	Execution Time (ms)
nAcqData( )	0.75
vFloatData( )	0.39
vFltrBw3( )	1.33
fftr( )	3.92
cvabs( )	0.55
maxv( )	0.16
vFindFloatPix( )	0.01
vFindTruGap( )	26.51
Total Processing Time	33.62

Note that the function *nServUART( )* does not take any considerable amount of time when there is no request pending for the system to send the data through the UART, otherwise it may take time in the tens of ms depending on the accepting performance of the personal computer to which it is connected to.

The integration time of the CCD spectrometer is fixed as the time of 8192 input clock pulses to it. For our 1.667 MHz input clock rate to the CCD spectrometer, the integration time is 4.92 ms. The total signal processing time of the EFPI sensor system, 33.62 ms, is many times larger than the integration time of the CCD spectrometer, and thus the system's real-time performance (i.e., the refresh rate of the output result) is limited mainly by the computational complexity (or the speed of the DSP processor) of the system. The refresh time period can only be a multiple of the integration time of the CCD spectrometer, and is 34.41 ms for our case, yielding a refresh rate of 29.06 per second.

6.2 Real-Time Performance of the FBG Sensor Systems

After loading all signal processing firmware and software of the FBG sensor system, as listed in Appendix A.2, on the zero wait-state SRAM memory of the 40 MHz 'C40 DSP processor, we measured the execution times for the individual functions. Given below are the function wise execution time of the real-time FBG optical fiber sensor system

Function Name	Execution Time (ms)
nAcqData( )	0.75
vFloatData( )	0.39
vFindPks( )	0.12
Total Processing Time	1.26

The total signal processing time of the FBG system (1.26 ms), is much less than the integration time of the CCD spectrometer (4.92 ms), and thus the system's real-time performance (i.e., the refresh rate of the output result) is limited not by the computational complexity of the system, but by the integration cycle of the CCD spectrometer. The refresh time can at the best be equal to the integration time of the CCD spectrometer, i.e., 4.92 ms, yielding a refresh rate of 203.45 per second for the FBG system.

6.3 Limitations of the Wavelength Modulated Optical Fiber Sensor Systems

The three main limiting factor in the real-time performance of our systems are: (a) the speed and the integration time of the linear array CCD spectrometer, (b) the speed of the analog-to-digital converter, and, (c) the speed of the TMS320C40 DSP processor. If the speed of all these three components can be increased by a factor, the refresh cycle time of our real-time systems can be improved with the same factor, assuming that the memory interfacing is still of zero wait-state for the DSP processor.

The limitations on the performance of the accuracy and the precision of the measurements are imposed by two factors: (a) the number of bits used for the analog-to-digital conversion (ADC) of the intensity amplitude, and (b) the total number of elements in the CCD spectrometer and also the spacing in wavelengths between the adjacent elements. Increasing the number of bits for the ADC and the number of elements for the CCD spectrometer the accuracy and the precision can be improved, but with a price of increased computation, which may slow the real-time performance.

Another limitation of our systems is that the SLED optical source has a output profile like a Gaussian curve. A perfectly white source having a constant output profile, though not practical

in reality, would provide a better performance in absence of the amplitude modulation of the optical intensity of the CCD spectrometer elements.

6.4 Noise Issues

The factors that mainly can introduce noise to the acquired data of the system are: (a) the noise due to finite-bit analog-to-digital conversion, (b) the noise due to the change of polarization of optics in the input/output fiber, (c) the noise due to the back-propagation of light to the SLED source after being reflected from the sensor interfaces, (d) the noise due to insertion of optical couplers and connectors, (e) the noise due to multiple reflections of optics inside the EFPI cavity, and, (f) the thermal noise.

By increasing the power of the optical output of the SLED source some of factors of the noise can be reduced, but increasing the optical power too much may saturate the spectrometer photodiodes, causing loss of information in the signal.

6.5 Comparisons of the Sensor Systems

We have seen that the signal processing hardware and the data acquisition firmware are the same for all the real-time systems of the EFPI, FBG, and the LPG sensors. The only difference among them is the real-time signal processing routines, and of course the sensor. The EFPI system entails very computationally intensive signal processing routines, while the FBG and LPG sensor systems requires routines which are very simple and not so intensive in computations.

The EFPI sensor system is the best for its wide dynamic range and high resistance to fragility, and it is inherently very resistant to the variations of temperature. The FBG and the LPG sensor systems are very sensitive to the variations of temperature, and for any measurable other than

temperature, these sensors must consider the noise introduced due to the variations of temperature that may occur.

The FBG and the LPG sensors are very suitable of multiplexing purposes due to their low insertion losses. But due signal processing advantages the FBG multiplexed systems might be the best, because of the fact that for each LPG sensor there are multiple dips in the transmission spectrum, which might cause confusion in tracking a single dip if several sensors are multiplexed together. The EFPI sensor is very unsuitable for multiplexed systems.

Chapter 7

Conclusions

We summarize our work and draw conclusions in this chapter. Suggestions for future enhancements on this work is made to lead someone to add to the performance of the wavelength modulated optical fiber sensor systems.

7.1 Summary and Conclusions

We develop the theory of a complete model of a wide-band extrinsic Fabry-Perot interferometric (EFPI) sensor system, along with the development of the real-time signal processing techniques of the signal response of the sensor. The developed model provides insight to the modulation of the signal response and their demodulation techniques, and enables one to simulate the signal response of a wavelength modulated EFPI sensor system for the purpose of research and analysis before building the entire system in hardware. For the first time, to our knowledge, in this work we model, design, build, and implement a complete stand alone optical fiber sensor system, of wavelength demodulation technique, which is capable of digitizing, acquiring, and processing the sensor signals over the whole spectrum of the wide optical source, and producing the precise and accurate sensing measurements, all in real-time. Texas Instruments TMS320C40 floating point DSP processor along with the analog-to-digital converter, the S1000 linear array spectrometer, the fast SRAM, the non-volatile SRAM, the UART, the watchdog timekeeper, and the Xilinx FPGAs are used to develop a stand-alone system to process the wavelength modulated optical fiber sensor signals in real-time. We develop in this work all the data acquisition and signal

processing firmware and software necessary for the real-time wavelength modulated optical fiber sensor systems.

A full suite of TCP/IP client/server software is developed to connect the stand alone wavelength modulated optical fiber sensor system to the Internet by using the system's modem capable UART interface. This enables one to monitor or acquire data from the system, from any remote locations using the Internet.

Among the wavelength modulated EFPI, the FBG, and the LPG optical fiber sensor systems, we found that the signal modulation of the EFPI sensor is the most complicated, and for signal demodulation it requires the most computationally intensive signal processing algorithms. But the EFPI system has also the best dynamic range of operations without having much sensitivity to the variations of temperature. For multiplexed sensing systems, the FBG technique is the best suitable, while it is very sensitive to temperature variations. The LPG sensor system is more sensitive in general than the FBG sensors, but the signal processing techniques are also a bit more complicated due to the fact that for each LPG sensor there are multiple dips in the transmission spectrum, each dip corresponding to the coupling of the fundamental guided mode to a different cladding mode

7.2 Future Enhancements

To increase the real-time performance of the wavelength modulated optical fiber sensor system, a faster CCD array than the Ocean Optics S1000 linear array spectrometer be developed, along with the development of a faster 16-bit analog-to-digital converter. A faster DSP processor will of course improve the real-time performance of the system significantly.

In developing the model of the wavelength modulated EFPI sensor system in Chapter 2, the derivation of the attenuation factor $f_a(g)$ as defined by Equation (2.3) is a very crude approximation. The loss effects in coupling the optical power back into the input/output fiber after the reflection at the sensing interface, as depicted in Figure 2.5, is too simple and is more appropriate for multi-mode fibers rather than for single-mode fibers. In single-mode fiber, light energy propagates not only through the fiber core, but a significant portion of light energy also propagates through the fiber cladding. The major limitations of Equation (2.3) are: (a) it does not consider the energy of light propagating through the cladding of the fiber, (b) it assumes that the light energy is uniformly distributed across the fiber core, which is not true for single-mode fibers, where energy is more concentrated at the center of the core, and, (c) it does not consider the effect of the hollow-core fiber which guides the input/output fiber and the reflector fiber of the EFPI sensor. As the factor $f_a(g)$ is a very important parameter of our model, a comprehensive research can be conducted to derive it accurately. The solutions to Maxwell's equations with the boundary conditions of the single-mode EFPI sensor may provide a very accurate derivation of the $f_a(g)$ factor.

An analytical proof is required to show that for any data set associated with a particular EFPI gap, the magnitude of the *discrete gap transformation* (DGT) of that data set is the maximum at that particular EFPI gap. The effect of the change of the parameters of the single-mode input/output fiber (e.g., the fiber diameter, or the fiber numerical aperture) as well as the effect of the variation of the optical source profile (e.g., the variation of the mean, or the spread of the Gaussian profile) on the DGT method should also be derived.

Not much theoretical analysis has yet been done on the LPG sensing technique, which is a fairly new technique. Opportunities exist to find out the analytic models for the LPG sensor system to relate the shift of wavelengths of the transmission loss peaks to the strain, temperature, or any other physical measurables.

Research should be conducted to find the effect of change of polarization on the model of the wavelength modulated EFPI sensor system. Also the effect of variation in refractive index with the change of wavelength can be found out theoretically for the wavelength modulated EFPI sensor system.

Research should be conducted to connect the stand alone wavelength modulated optical fiber sensor system to directly connect to the Internet by using the Ethernet chip-set on the board and by using an operating TCP/IP software kernel imbedded in the system, rather than via a personal computer as we implemented. Note that all the hardware components and the software already exists in the market to implement the imbedded system. The system cost can be reduced dramatically by implementing the Internet connection through the imbedded system, as the personal computer is no longer necessary for an imbedded system.

Appendix A

The Real Time Signal Processing Firmware and Software

A.1 Real-Time Signal Processing Routines for the EFPI Sensor System

```
/*****************************************************************
Function Prototype:
void main (void)

Author and Date Started: Shah Musa, 3/9/97

Description:
The main() program calls all the initialization, data acquisition, and signal
processing functions.  The functions could either be in C, or in C4x assembly
language.

Method:
The InitC4x() function initializes all the necessary registers of the
TMS320C4x DSP processor.  The InitUART() function initializes all the
necessary registers of the UART for communicating with the comport or
computers.  The CalbPxls(flaot *) function calibrates the 1100 CCD pixels to
wavelengths and the table is loaded at the input pointer.  It uses the Ocean
Optics CCD spectrometer calibration coefficients.  Every individual CCD
spectrometer, having a unique SIE number has its own calibration coefficients.
(Note that the FFT calibration coefficients are not the CCD calibration
coefficients.)  The AcqData(int iaData[],DMA3_DEST,DMA4_DEST) acquires the
1100 data set at the given pointer iaData.  There must be at least 275 (0x44c)
memory locations both at iaDMA3_DEST and iaDMA4_DEST.

The rest of the functions are for the real-time signal processing for the EFPI
wavelength modulated optical fiber sensor.

Any new functions can easily be added in main() at it's proper place,
if necessary.


Input Parameters and Ranges:
None.

Called By:
None.

*****************************************************************/
```

```
#include <stdlib.h>
#include <math.h>

/* If the stack is small, then large arrays must not be kept as local
variables.  Local variables are placed on stack for processing.  The following
variables are kept out of main() for monitoring from the 'watch' window. */

float
      faData[1024],
      faFourPiByLamda[1024],
      faFltrdData[1024],
      faFFTMag[1024];
float
      fGapEst,
      fTruGap,
      fMaxPix,
      fMaxVal;
short
      nStatus;
int
      iIndexOfMax,
      iaData[1100],
      iaDMA3_DEST[275],
      iaDMA4_DEST[275];
typedef struct
      {
      float r;
      float i;
      } complex;
complex
      FFTData[1024];

void main(void)
{

      float
            fMaxPix,fMaxVal;
      int
            i;
      float
            C2=-5.0963e-6,
            C1=1.8196,
            C0=-2.354;
            /* FFT calibration coefficients for CCD of SIE687 */

      vInitC4x();
      vInitUART();
      vCalbPxls(faFourPiByLamda);

      for (;;){
            nStatus = nAcqData(iaData,iaDMA3_DEST,iaDMA4_DEST);
                  /* nStatus = 0 , if nAcqData() is successful */
            nStatus = nServUART(iaData);
                  /* nStatus = 0 , if nServUART() sends data to UART at the
                  request of server */
            vFloatData(iaData+16,faData);
                  /* Type conversion, converts data from int to float */
```

```
                /* Actual data starts 16 pixel later */
            vFltrBw3(faData,faFltrdData,1024);  /* High-Pass filter */
            fftr(faFltrdData,FFTData,1024,0);  /* SigTar function */
            cvabs(FFTData,2,faFFTMag,1,512);  /* VecTar function */
            maxv(faFFTMag,1,&fMaxVal,&iIndexOfMax,512); /* VecTar function */
            vFindFloatPix(iIndexOfMax,faFFTMag,&fMaxPix);
                /* Gaussian interpolation */
            fGapEst = C0 + C1*fMaxPix + C2*fMaxPix*fMaxPix;
            vFindTruGap(fGapEst,&fTruGap,faData,faFourPiByLamda);
      }
}



*************************************************************************
* Function Prototype:
* void vInitC4x(void),
* called from C environment.
*
* Author and Date Started: Shah Musa, 3/1/97
*
* Description:
* The function vInitC4x( ) initializes all the necessary CPU and peripheral
* bus registers of the C4x processor.  After setting the timer TCLK0 as an
* output clock, vInitC4x( ) waits for a while for the timer to be stable,
* before acquiring any data from the CCD.  TCLK0 is used as the input clock
* to the CCD.  The global memory interface control register (GMICR) and the
* local memory interface control register (LMICR) are set properly with
* appropriate strobe sectioning and wait state conditions for the memory
* sections.
*
* Input Parameters and Ranges:
* None.
*
* Returns:
* None.
*
* Called By:
* main ().
*
*************************************************************************

DIER        .word 0h          ;DMA interrrupt enable register
IIER        .word 0h          ;Internal interrupt enable register
IIFR        .word 00002202h   ;Flag register
PERIPH      .word 00100000h   ;Base address of the peripheral registers
TIM0CTL     .word 000003C1h   ;Timer 0 control Register (1 MHz CLK) (20h)
TIM0PRD     .word 00000003h   ;Timer 0 period register (28h)
GMICR       .word 3E39D952h   ;Global memory interface control register (0h)
LMICR       .word 3E39F852h   ;Local memory interface control register (4h)

            .def  _vInitC4x

_vInitC4x:

      LDI   0A000h,ST         ;GIE=1,SET COND=1
```

```
      LDI   @DIER,DIE
      LDI   @IIER,IIE
      LDI   @IIFR,IIF
      LDI   @PERIPH,AR7
      LDI   @GMICR,R0
      STI   R0,*+AR7(0h)      ;Init global memory interface control register
      LDI   @LMICR,R0
      STI   R0,*+AR7(4h)      ;Init local memory interface control register
      LDI   @TIM0PRD,R0
      STI   R0,*+AR7(28h)     ;Set timer 0 period register
      LDI   @TIM0CTL,R0
      STI   R0,*+AR7(20h)     ;Set timer 0 control

      ;wait for a while for the TCLK0 output clock signal to be stable
      LDHI  06h,R0
      OR    0FFFFh,R0;(This wait is very crucial for correct acquisition)
HERE
      SUBI  1h,R0
      BLE   THERE
      BR    HERE
THERE
      RETS


*************************************************************************
* Function Prototype:
* void vInitUART(void),
* called from C environment.
*
* Author and Date Started: Tracy Cramer & Shah Musa, 3/1/97
*
* Description:
* The function vInitUART( ) initializes all the necessary registers of the
* UART and also sets a baud rate for the UART.  Note that three NOPs are
* required after any read or write to the UART.
*
* Input Parameters and Ranges:
* None.
*
* Returns:
* None.
*
* Called By:
* main ().
*
*************************************************************************

            .def  _vInitUART
UARTADDR    .word 00300000h   ; Address of RS232

IER         .set  4     ; Interrupt Enable Register (changed from 1)
IIR         .set  2     ; Interrupt Identification Register(read only)
FCR         .set  2     ; FIFO Control Register(write only)
LCR         .set  6     ; Line Control Register (changed from 3)
MCR         .set  1     ; MODEM Control Register (changed from 4)
LSR         .set  5     ; Line Status Register
MSR         .set  3     ; MODEM Status Register (changed from 6)
```

```
SCR         .set  7     ; Scratch Register
DLL         .set  0     ; Divisor Latch Least(significant byte)
DLM         .set  4     ; Divisor Latch Most(significant byte)(changed from 1)
DLAB        .set  80h   ; Divisor Latch Access Bit value in the LCR

* Baud rate constants for 18.432MHz clock and 38400bps
BAUD_HI     .set  00
BAUD_LO     .set  30

_vInitUART:

* We initialize all of the UART's registers by reading them.
* This clears any flags and removes any previous characters that
* were entered.
      LDI         @UARTADDR,AR0     ; Load RS232 base address to AR0
      LDI         *AR0,R0           ; Clear the receive register
      NOP
      NOP
      NOP
      LDI         *+AR0(IER),R0     ; Clear all registers by reading them
      NOP
      NOP
      NOP
      LDI         *+AR0(IIR),R0
      NOP
      NOP
      NOP
      LDI         *+AR0(FCR),R0
      NOP
      NOP
      NOP
      LDI         *+AR0(LCR),R0
      NOP
      NOP
      NOP
      LDI         *+AR0(MCR),R0
      NOP
      NOP
      NOP
      LDI         *+AR0(LSR),R0
      NOP
      NOP
      NOP
      LDI         *+AR0(MSR),R0
      NOP
      NOP
      NOP
      STI         0h,*+AR0(SCR)     ; Clear scratch register
      NOP
      NOP
      NOP
      STI         2,*+AR0(MCR)      ; Enable RTS
      NOP
      NOP
      NOP

* The following code sets the baud rate (38400bps) for the UART
```

```
        LDI         DLAB,R0
        STI         R0,*+AR0(LCR)     ; Set DLAB
        NOP
        NOP
        NOP
        LDI         BAUD_HI,R0
        STI         R0,*+AR0(DLM)     ; Set the divisor latch MSB
        NOP
        NOP
        NOP
        LDI         BAUD_LO,R0
        STI         R0,*AR0           ; Set the divisor latch LSB
        NOP
        NOP
        NOP
        STI         3h,*+AR0(LCR)     ; Reset DLAB and set 8,none,1
        NOP
        NOP
        NOP
        RETSU



/************************************************************************
Function Prototype:
void vCalbPxls(float *pfFourPiByLamda)

Author and Date Started: Shah Musa, 4/12/97

Description:
The function, vCalbPxls(float *pfFourPiByLamda), initializes the array of 1024
floating point values with the w (4*pi/lamda) vlaues of the 1024 elements
(starting at element 17) of the Ocean Optics CCD. The first 16 CCD elements
are not photodiodes.  Note that the conversion coefficients (fCoeff1, fCoeff2,
and fIncept) are different for different CCDs.  Make sure to put the right
coefficients using the SIE# of the CCD being used.  All the lamda values are
in nm.


Input Parameters and Ranges:

float *pfFourPiByLamda:
pfFourPiByLamda must be a pointer to an array of 1024 floating point values.

Returns:
None.

Called By:
main ().

************************************************************************/


void vCalbPxls(float *pfFourPiByLamda)
{
```

```
      float
            fCoeff1 = 0.24256929,
            fCoeff2 = -4.176e-5,
            fIncept = 744.837472;
            /* Coefficients for the CCD of SIE687 */

float
      fLamda,
      fPi = 3.141592654;
int   i,
      iPix,
      iPixOffset=16;
            /* iPixOffset is the index of the first photodiode element of the
            CCD array in the 1100 acquired data vector */

      for (i=0; i<1024; i++)
      {
            iPix = i + iPixOffset;
            fLamda = fIncept + fCoeff1*iPix + fCoeff2*iPix*iPix;
                  /* fLamda in nm */
            *(pfFourPiByLamda+i) = (4000*fPi)/fLamda;
      }
}



*************************************************************************
* Function Prototype:
* short nAcqData(int *iaData,int *iaDMA3_DEST,int *iaDMA4_DEST),
* called from C environment.
*
* Author and Date Started: Shah Musa, 11/20/96
*
* Description:
* The function, nAcqData(int *iaData,int *iaDMA3_DEST,int *iaDMA4_DEST),
* acquires 1100 integer data set from the CCD spectrometer, and saves them
* in the array pointed by iaData.
*
* Method:
* DMA3 and DMA4 are initialized, after reseting, to take 275 word of data each
* from comport 3 and comport 4, at locations pointed by iaDMA3_DEST and
* iaDMA4_DEST respectively.  (iaDMA3_DEST holds the least significant bytes
* of the 1100 CCD data as full 32-bit words in consecutive (1100/4 = 275)
* memory locations.  iaDMA4_DEST holds the most significant bytes of the
* 1100 CCD data as full 32-bit words in consecutive (1100/4 = 275) memory
* locations.)  After enabling the CCD spectrometer
* and the DMA interrup enable, the CPU waits in a loop (WAITLOOP) until the
* data set is acquired completely.  Then it disables both the CCD enable and
* the DMA interrupt enable, and calls the UNMINGLE subroutine, after proper
* register  initializations, to unmingle the byte mingled data in 1100
* sign-extented 16-bit integer numbers at location pointed by iaData.
*
* Note that, in the WAITLOOP it watis only a predetermined amount of time
* (determined by WAITCOUNT), and if the complete data set is not acquired
* within that time, the routine nAcqData() returns with an error code of -1,
* otherwise if successful it returns with a value of 0.
```

```
*
*Input Parameters and Ranges:
*
* int *iaData:
* iaData points to the integer array where the 1100 acquired data set
* be saved.
*
* int *iaDMA3_DEST:
* iaDMA_DEST points to the integer array where the least significant bytes
* of the 1100 CCD data is acquired as full 32-bit words in consecutive
* 1100/4=275 memory locations.
*
* int *iaDMA3_DEST:
* iaDMA_DEST points to the integer array where the most significant bytes
* of the 1100 CCD data is acquired as full 32-bit words in consecutive
* 1100/4=275 memory locations.
*
* Returns:
* A short integer, 0 if succesful, and -1 if data acquisition is failed.
*
* Called By:
* main ().
*
************************************************************************

            .def  _nAcqData

* Initilize all necessary CPU registers
DIER        .word 00104000h   ;DMA3-DMA4 read through ICRDY3-ICRDY4 interrupts

* Initilize all necessary internal peripheral registers
DMA3_CTL    .word 00C00047h         ;START=11, TCC=0, SYNC_MODE=01, and,
                                    ;TRANSFER_MODE=01, DMA_PRI=11 (D0h)
DMA3_SRC    .word       00100071h   ;source reg (D1h)
DMA3_SRCI   .word       0h          ;source index register value (D2h)
DMA3_COUNT  .word       275         ;counter register value, 275*4=1100 (D3h)
;DMA3_DEST  .word       80005000h   ;destination address (raw LS data) (D4h)
DMA3_DESTI  .word       1h          ;destination index register value (D5h)
DMA4_CTL    .word       00C00047h   ;START=11, TCC=1, SYNC_MODE=01, and,
                                    ;TRANSFER_MODE=01, DMA_PRI=11 (E0h)
DMA4_SRC    .word       00100081h   ;source reg (E1h)
DMA4_SRCI   .word       0h          ;source index register value (E2h)
DMA4_COUNT  .word       275         ;counter register value, 275*4=1100 (E3h)
;DMA4_DEST  .word       80005200h   ;destination address (raw MS data) (E4h)
DMA4_DESTI  .word       1h          ;destination index register value (E5h)

* Initilize other variables required
PERIPH      .word       00100000h   ;Starting address of internal peripheral-
                                    ;bus memory
WAITCOUNT   .word       003FFFFFh   ;The maximum delay allowed for the data
                                    ;acquisition

_nAcqData:

            PUSH  DIE         ;Save DMA Internal Interrup register
            PUSH  IIF         ;Save Internal Interrupt Flag register
            PUSH  AR0         ;Save AR0
```

```
            PUSH  AR1         ;Save AR1
            PUSH  AR2         ;Save AR2
            PUSH  AR3         ;Save AR3
            PUSH  AR4         ;Save AR4
            PUSH  AR5         ;Save AR5
            PUSH  AR6         ;Save AR6
            PUSH  AR7         ;Save AR7, points to peripheral registers
            PUSH  R1          ;Save the lower 32 bits of R1
            PUSH  R2          ;Save the lower 32 bits of R2

            ANDN  0400h,IIF   ;Put IIOF2 output to 0,
                              ;IIOF2 is the CCD enable input


* Reset C40 COM3 and COM4 FIFOs
RST_FIFO3
            LDI   *+AR7(070h),R0    ;Load CPCR3 to R0
            AND   01E00h,R0         ;Input level is bit 9,10,11,12
            BZ    RST_FIFO4
            LDI   *+AR7(071h),R0    ;Read input FIFO
            BR    RST_FIFO3
RST_FIFO4
            LDI   *+AR7(080h),R0    ;Load CPCR4 to R0
            AND   01E00h,R0         ;Input level is bit 9,10,11,12
            BZ    DONE_FIFO4
            LDI   *+AR7(081h),R0    ;Read input FIFO
            BR    RST_FIFO4
DONE_FIFO4

            STI    0h,*+AR7(0D0h)   ;Reset DMA 3 control register
            STI    0h,*+AR7(0E0h)   ;Reset DMA 4 control register

* set DMA channel 3 to read data from Com Port 3
            LDI    @DMA3_CTL,R0     ;Set DMA 3 control register
            STI    R0,*+AR7(0D0h)
            LDI   @DMA3_SRC,R0      ;Set DMA 3 source register
            STI   R0,*+AR7(0D1h)
            LDI   @DMA3_SRCI,R0     ;Set DMA 3 source index register
            STI   R0,*+AR7(0D2h)
            LDI   @DMA3_COUNT,R0    ;Set DMA 3 counter register
            STI   R0,*+AR7(0D3h)

            LDI   SP,AR0;
            LDI   *-AR0(14),R0      ;Load the DMA3_DEST argument to R0.
                                    ;(12 PUSHes + 2 = 14)
                                    ;R0 points to LS-byte of data.
            ;LDI  @DMA3_DEST,R0
            STI   R0,*+AR7(0D4h)    ;Set DMA 3 destination register
            LDI   @DMA3_DESTI,R0    ;Set DMA 3 destination index register
            STI   R0,*+AR7(0D5h)

* set DMA channel 4 to read data from Com Port 4
            LDI   @DMA4_CTL,R0      ;Set DMA 4 control register
            STI   R0,*+AR7(0E0h)
            LDI   @DMA4_SRC,R0      ;Set DMA 4 source register
            STI   R0,*+AR7(0E1h)
            LDI   @DMA4_SRCI,R0     ;Set DMA 4 source index register
```

```
            STI   R0,*+AR7(0E2h)
            LDI   @DMA4_COUNT,R0    ;Set DMA 4 counter register
            STI   R0,*+AR7(0E3h)

            LDI         SP,AR0
            LDI   *-AR0(15),R0      ;Load the DMA4_DEST argument to R0.
                                    ;(12 PUSHes + 3 = 15)
                                    ;R0 points to MS-byte of data.
            ;LDI  @DMA4_DEST,R0
            STI   R0,*+AR7(0E4h)    ;Set DMA 4 destination register
            LDI   @DMA4_DESTI,R0    ;Set DMA 4 destination index register
            STI   R0,*+AR7(0E5h)

            LDI   @DIER,DIE         ;Enable ICRDY3-ICRDY4 read sync
                                    ;(DIE is initialized after starting DMA)
            OR    0400h,IIF         ;Put IIOF2 output to 1 (enable CCDs)

* Check and wait until a fixed time, for the 1100 data  be acquired
            LDI   @WAITCOUNT,R2     ;R2 holds the maximum waitcount
WAITLOOP
            LDI   *+AR7(0D3h),R0    ;Load DMA3 transfer counter in R0
            LDI   *+AR7(0E3h),R1    ;Load DMA4 transfer counter in R1
            OR    R0,R1
            BZ    DATA_ACQD
            SUBI  1,R2              ;Decrement the waitcount by 1
            BNN   WAITLOOP          ;Go to WAITLOOP if waitcount is not yet
                                    ;counted all down
            LDI   0h,DIE            ;Disable ICRDY3-ICRDY4 read sync
            ANDN  0400h,IIF         ;Put IIOF2  output to 0 (disable CCDs)
            LDI   -1,R0
            BU    NOT_ACQD          ;Return with -1, if data not acquired
DATA_ACQD
            LDI   0h,DIE            ;Disable ICRDY3-ICRDY4 read sync
            ANDN  0400h,IIF         ;Put IIOF2  output to 0 (disable CCDs)

* Call UNMINGLE subroutine after proper register initializations
            LDI   SP,AR0;
            LDI   *-AR0(14),AR4     ;Load the DMA3_DEST argument to AR4
                                    ;(12 PUSHes + 2 = 14)
                                    ;DMA3_DEST points to LS-bytes of data
            LDI   *-AR0(15),AR5     ;Load the DMA4_DEST argument to AR5
                                    ;(12 PUSHes + 3 = 15)
                                    ;DMA4_DEST points to MS-bytes of data
            LDI   *-AR0(13),AR6     ;Load the iaData[] argument to AR6
                                    ;(12 PUSHes + 1 = 13)
                                    ;AR6 points to the unmingled data
            CALL  UNMINGLE
            NOP
            LDI   0,R0        ;return with 0, when data is acquired.
NOT_ACQD
            POP   R2          ;Restore the lower 32 bits of R2
            POP   R1          ;Restore the lower 32 bits of R1
            POP   AR7         ;Restore AR7
            POP   AR6         ;Restore AR6
            POP   AR5         ;Restore AR5
            POP   AR4         ;Restore AR4
            POP   AR3         ;Restore AR3
```

```
            POP   AR2         ;Restore AR2
            POP   AR1         ;Restore AR1
            POP   AR0         ;Restore AR0
            POP   IIF         ;Restore IIF
            POP   DIE         ;Restore DIE
            RETS


*************************************************************************
* Function Prototype:
* UNMINGLE, called from C4x assembly environment.
*
* Author and Date Started: Shah Musa, 11/20/96
*
* Description:
* Subroutine UNMINGLE unmingles the raw data (saved by the DMA3 and DMA4),
* and converts them in 16-bit sign-extented integer values.
*
*
* Input Parameters and Ranges:
*
* It is assumed that:
* (a) AR4 already points to the address of least significant bytes
* of the 1100 CCD data as full 32-bit words in consecutive (1100/4 = 275)
* memory locations
* (a) AR5 already points to the address of most significant bytes
* of the 1100 CCD data as full 32-bit words in consecutive (1100/4 = 275)
* memory locations
* (c) AR6 already holds the address pointing to the destination of the
* final integer data values.
*
* Returns:
* None.
*
* Called By:
* nAcqData().
*
*************************************************************************

UNMINGLE:

;Save all the registers being used and modified
            PUSH AR0          ;Save AR0
            PUSH AR1          ;Save AR1
            PUSH  R0          ;Save the lower 32 bits of R0
            PUSH  R1          ;Save the lower 32 bits of R1
            PUSH  R2          ;Save the lower 32 bits of R2
            PUSH  R3          ;Save the lower 32 bits of R3
            PUSH  R4          ;Save the lower 32 bits of R4
            PUSH  R5          ;Save the lower 32 bits of R5
            PUSH  R6          ;Save the lower 32 bits of R6
            PUSH  R7          ;Save the lower 32 bits of R7
            PUSH  R8          ;Save the lower 32 bits of R8
            PUSH  R9          ;Save the lower 32 bits of R9

            LDI   AR6,AR0     ;Copy the address of final destination to AR0
            LDI   AR6,AR1     ;Copy the address of final destination to AR1
```

```
;Start unmingling the byte-mingled data
;(The Com Port keeps the first received byte as the LS-byte of the word)
            LDI   1100/4-1,RC       ;Load the repeat counter
            NOP
            RPTB  UNPACK
            LDI   *AR5++(1),R5      ;R5 has the 4 MS bytes of 4 data values
      ||    LDI   *AR4++(1),R4      ;R4 has the 4 LS bytes of 4 data values
            LDI   0FFh,R6           ;Load 0000 00FFh to R6
            AND3  R5,R6,R8
            AND3  R4,R6,R7
            LDI   0h,R0
            LWL2  R7,R0
            LWL3  R8,R0             ;First data is in the MS 16 bits of R0
            LH1   R0,R0             ;First data is in the LS 16 bits of R0 and
                                    ;sign extended
            STI   R0,*AR6++(1)

            LDI   0FF00h,R6
            AND   0FF00h,R6         ;Load 0000 FF00h to R6
            AND3  R5,R6,R9
            LDI   0h,R8
            LWR1  R9,R8             ;Shifted right to help merge
            AND3  R4,R6,R7
            LDI   0h,R1
            LWL1  R7,R1
            LWL3  R8,R1             ;Second data is in the MS 16 bits of R1
            LH1         R1,R1       ;Second data is in the LS 16 bits of R1
                                    ;and sign extended
            STI   R1,*AR6++(1)

            LDHI  0FFh,R6           ;Load 00FF 0000h to R6
            AND3  R5,R6,R9
            LDI   0h,R8
            LWR2  R9,R8             ;Shifted right to help merge
            AND3  R4,R6,R7
            LDI   0h,R2
            LWL0  R7,R2
            LWL3  R8,R2             ;Third data is in the MS 16 bits of R2
            LH1         R2,R2       ;Third data is in the LS 16 bits of R2 and
                                    ;sign extended
            STI   R2,*AR6++(1)

            LDHI  0FF00h,R6         ;Load FF00 0000h to R6
            AND3  R5,R6,R9
            LDI   0h,R8
            LWR3  R9,R8             ;Shifted right to help merge
            AND3  R4,R6,R7
            LDI   0h,R3
            LWR1  R7,R3
            LWL3  R8,R3             ;Fourth data is in the MS 16 bits of R3
            LH1         R3,R3       ;Fourth data is in the LS 16 bits of R3
                                    ;and sign extended
UNPACK      STI   R3,*AR6++(1)
            NOP

;Restore all the stacked registers
```

```
            POP   R9          ;Restore the lower 32 bits of R9
            POP   R8          ;Restore the lower 32 bits of R8
            POP   R7          ;Restore the lower 32 bits of R7
            POP   R6          ;Restore the lower 32 bits of R6
            POP   R5                ;Restore the lower 32 bits of R5
            POP   R4                ;Restore the lower 32 bits of R4
            POP   R3                ;Restore the lower 32 bits of R3
            POP   R2                ;Restore the lower 32 bits of R2
            POP   R1                ;Restore the lower 32 bits of R1
            POP   R0                ;Restore the lower 32 bits of R0
            POP   AR1               ;Restore AR1
            POP   AR0               ;Restore AR0
            RETS



*************************************************************************
* Function Prototype:
* short nServUART(int *iaData),
* called from C environment.
*
* Author and Date Started: Shah Musa, Tracy Cramer 3/1/97
*
* Description:
* The function nServUART( ), if called from the main( ),
* checks if there is any request for sending the 1100 integer data set
* through the UART.  If it finds the request 'S' it sends the data
* with the preamble (123456789) and the postamble (987654321).
* Otherwise it returns from the subroutine doing nothing.
* Note after every read or write of the UART needs 3 NOPs
* to be followed by.
*
* Input Parameters and Ranges:
*
* int *iaData:
* iaData points the 1100 integer data set to be sent through the UART.
*
* Returns:
* It returns with 0 if it finds a request and sends the data, otherwise it
* returns with -1.
*
* Called By:
* main ().
*
*************************************************************************

            .def        _nServUART
UARTADDR    .word       00300000h   ; Address of RS232
DATAADDR    .word       80004000h   ; Address of 1100 data location
BEGINING    .word       123456789;  ; Preamble of the Data set to be sent
ENDING      .word       987654321;  ; Postamble of the Data set to be sent

MCR         .set  1     ; MODEM Control Register (changed from 4)
LSR         .set  5     ; Line Status Register
```

```
MSR         .set  3     ; MODEM Status Register (changed from 6)

_nServUART:
      PUSH  AR0
      PUSH  AR1
      PUSH  R1
      PUSH  RC

      LDI   @UARTADDR,AR0     ; Load RS232 address to AR0
      ;STI  0h,*+AR0(MCR)     ; Disables the RTS bit
      NOP
      NOP
      NOP
      LDI   *+AR0(LSR),R0     ; Check the received character bit of PC16550
      NOP
      NOP
      NOP
      AND         1h,R0
      BZ          RETURNM     ; If the result is zero there are no
                              ; characters in the buffer
      LDI         *AR0,R0     ; Else, we have a character.  Load it into R0
      NOP
      NOP
      NOP
      AND   0FFh,R0
      SUBI3 'S',R0,R1         ; Check to see if the character is an 'S'(53h);
      BZ    SENDDATA
      LDI   -1,R0
      BU    RETURNM           ; If not, return with -1

      ; If the character is an 'S', go ahead and send the data
SENDDATA
      LDI   @BEGINING,R1      ; Send the preamble first
      CALL  XMIT_WORD

      LDI   SP,AR0
      LDI   *-AR0(5),AR1      ; Load the iaData[] argument to AR1
      LDI   1100-1,RC         ; Load the repeat counter
      RPTB  SENDIT
      LDI   *AR1++(1), R1     ; Puts data in R1
      CALL  XMIT_WORD
      NOP
SENDIT
      LDI   @ENDING,R1
      CALL  XMIT_WORD
      NOP
      LDI   0h,R0             ; Return with 0 if data is sent
RETURNM
      STI   2h,*+AR0(MCR)     ; Enable the RTS bit
      NOP
      NOP
      NOP
      POP   RC
      POP   R1
      POP   AR1
      POP   AR0
      RETSU
```

```
*************************************************************************
* Function Prototype:
* XMIT_WORD,
* called from C4x assembly environment.
*
* Author and Date Started: Tracy Cramer & Shah Musa, 3/1/97
*
* Description:
* The function XMIT_WORD transmits the contents of R1 to the UART.  It
* first reorders the bytes of R1 to send the MS byte first.
*
* Input Parameters and Ranges:
* It is assumed that the word be sent is already in R1 before calling this
* function.
*
* Returns:
* None.
*
* Called By:
* vServUART().
*
*************************************************************************

XMIT_WORD:
      PUSH  R0
      PUSH  R2
      PUSH  R3
      PUSH  R4
      PUSH  AR0

* Reverse the contents of R1 (i.e. R1=B3B2B1B0h becomes R1=B0B1B2B3h),
* to send the MS byte first.
      LDI   0h,R0       ; R0=0000 0000h
      LDI   0h,R2       ; R2=0000 0000h
      LHU0  R1,R2       ; R2=0000 B1B0h, R1=B3B2 B1B0h
      LSH   -8,R2,R0    ; R0=0000 00B1h
      AND   0FFh,R2     ; R2=0000 00B0h
      LWL1  R2,R0       ; R0=0000 B0B1h
      LHU1  R1,R2       ; R2=0000 B3B2h
      LSH   -8,R2,R1    ; R1=0000 00B3h
      AND   0FFh,R2     ; R2=0000 00B2h
      LWL1  R2,R1       ; R1=0000 B2B3h
      LWL2 R0,R1        ; R1=B0B1 B2B3h (reversed)
      ; End of reversing the contents of R1

      LDI   @UARTADDR,AR0
      LDI   -8,R3
      LDI   4,R4
LP0:
      LDI   *+AR0(LSR),R0     ; Read the Line Status Register
      NOP
      NOP
      NOP
```

```
      AND   20h,R0            ; If the THRE bit is high we are ok to put the
                              ; byte in the buffer.
      BZ          LP0
LP1:
      LDI   +AR0(MSR),R0      ; Read the MODEM Status Register
      NOP
      NOP
      NOP
      AND   10h,R0            ; If the CTS bit is high we are clear to send
                              ; the character
      BZ    LP1
      STI   R1,*AR0           ; Sends the eight LSB
      NOP
      NOP
      NOP
      LSH   R3,R1             ; Shifts R1 eight bits right (see header)
      SUBI  1h,R4             ; Decrement counter
      BNZ   LP0               ; Transmit four bytes

      POP   AR0
      POP   R4
      POP   R3
      POP   R2
      POP   R0
      RETSU



*************************************************************************
* Function Prototype:
* void vFloatData(int *piData,float *pfData);
*
* Author and Date Started: Shah Musa, 2/13/97
*
* Description:
* vFloatData(int *piData,float *pfData) converts 1024 integer data pointed
* by piData to floating point values and stores them to memory pointed
* by pfData.
*
* Input Parameters and Ranges:
*
* int *piData:
* The array of 1024 integer data to be converted is pointed by piData.
*
* float *pfData:
* The memory location where the converted floating point values be stored
* is pointed by pfData.
*
* Returns:
* None
*
* Called By:
* main ().
*
*************************************************************************/
```

```
   .def    _vFloatData

_vFloatData:
    PUSH    AR0
    PUSH    AR1
    PUSH    R0
    PUSH    R1
    LDI     SP,AR0
    LDI     *-AR0(5),AR1 ;AR1 is a pointer to the data we need to convert
    LDI     *-AR0(6),AR0 ;AR0 contains the converted data

    LDI     1024-1,RC ;Init repeat counter
    RPTB    FLTIT ;Repeat RC+1 times
    LDI     *AR1++(1),R0
    FLOAT   R0,R1
FLTIT:
    STF     R1,*AR0++(1)
    POP     R1
    POP     R0
    POP     AR1
    POP     AR0
    RETS



/************************************************************************
Function Prototype:
void vFltrBw3(float x[], float y[], int iNumPts)

Author and Date Started: Shah Musa, Kevin Shinpaugh, 2/13/97

Description:
vFltrBw3(float x[], float y[], int iNumPts) is a 3rd order butter worth high
pass filter with a normalized cutoff frequency of fc=0.03.  x[] points to the
input data, and y[] points to the filtered output data.  The length of the
output array is equal to the length of the input array.

Input Parameters and Ranges:

float x[]:
The array of floating point data values, to be filtered, is pointed by x.

float y[]:
y points to the array of filtered data values.

int iNumPts:
iNumPts is the length of the input array of x[].

Returns:
None

Called By:
main ().

************************************************************************/
```

```
void vFltrBw3(float x[], float y[], int iNumPts)
{
      float b[4] = {0.91002543068616,
                   -2.73007629205848,
                    2.73007629205848,
                   -0.91002543068616};
      float a[4] = {1.00000000000000,
                   -2.81157367732469,
                    2.64048349277834,
                   -0.82814627538626};
      int i;

      y[0] = b[0]*x[0];
      y[1] = b[0]*x[1] + b[1]*x[0] - a[1]*y[0];
      y[2] = b[0]*x[2] + b[1]*x[1] + b[2]*x[0] - a[1]*y[1] - a[2]*y[0];

      for (i=3; i<iNumPts; i++)
      {
            y[i] = b[0]*x[i] + b[1]*x[i-1] + b[2]*x[i-2] + b[3]*x[i-3]
                   - a[1]*y[i-1] - a[2]*y[i-2] - a[3]*y[i-3];
      }
}



*************************************************************************
* Function Prototype:
* void vFltrBw3(float x[], float y[], int iNumPts)
* (Implemented in C4x Assembly)
*
* Author and Date Started: Shah Musa, Tracy Cramer 4/12/97
*
* Description:
* vFltrBw3(float x[], float y[], int iNumPts) is a 3rd order butter worth
* high pass filter with fc=0.03.  x[] points to the input data, and y[]
* points to the filtered output data.  The length of the output array
* is equal to the length of the input array.
*
* Input Parameters and Ranges:
*
* float x[]:
* The array of floating point data values, to be filtered, is pointed by x.
*
* float y[]
* y points to the array of filtered data values.
*
* int iNumPts:
* iNumPts is the length of the input array of x[].
*
* Returns:
* None
*
* Called By:
* main (), (afss.c)
```

```
*******************************************************************************

    .def    _vFltrBw3

b0  .float     0.91002543068616
b1  .float    -2.73007629205848
b2  .float     2.73007629205848
b3  .float    -0.91002543068616
a2  .float    -2.81157367732469
a3  .float     2.64048349277834
a4  .float    -0.82814627538626

_vFltrBw3:
    PUSH    AR0
    PUSH    AR1
    PUSH    AR2
    PUSH    AR3
    PUSH    R0
    PUSH    R1
    PUSH    R2
    PUSH    R3
    PUSH    R4
    PUSH    R5
    PUSH    R6
    PUSH    R7 ; general purpose register
    PUSH    R8 ; general purpose register
    PUSH    R9 ; general purpose register

    LDI     SP,AR3
    LDI     *-AR3(15),AR0 ; Pointer to input array
    LDI     *-AR3(16),AR1 ; Pointer to output array
    LDI     *-AR3(17),AR2  ; Number of points

*   Load constants for quicker access
    LDF     @b0,R0
    LDF     @b1,R1
    LDF     @b2,R2
    LDF     @b3,R3
    LDF     @a1,R4
    LDF     @a2,R5
    LDF     @a3,R6

*   y[0] = b[0]*x[0];
    MPYF3   R0,*AR0,R7
    STF     R7,*AR1

*   y[1] = b[0]*x[1] + b[1]*x[0] - a[1]*y[0];
    MPYF3   R0,*+AR0(1),R7
    MPYF3   R1,*AR0,R8
    ADDF    R7,R8
    MPYF3   R4,*AR1,R7
    SUBF    R7,R8
    STF     R8,*+AR1(1)

*   y[2] = b[0]*x[2] + b[1]*x[1] + b[2]*x[0] - a[1]*y[1] - a[2]*y[0];
    MPYF3   R0,*+AR0(2),R7
    MPYF3   R1,*+AR0(1),R8
```

```
    ADDF    R7,R8
    MPYF3   R2,*AR0,R7
    ADDF    R7,R8
    MPYF3   R4,*+AR1(1),R7
    SUBF    R7,R8
    MPYF3   R5,*AR1,R7
    SUBF    R7,R8
    STF     R8,*+AR1(2)

*   for (i=0; i<iNumPts-3; i++)
    SUBI    4,AR2
    LDI     AR2,RC
    RPTB    FILTER_LOOP

*   y[i+3] = b[3]*x[i] + b[2]*x[i+1] + b[1]*x[i+2] + b[0]*x[i+3]
*             - a[2]*y[i] - a[1]*y[i+1] - a[0]*y[i+2];
    MPYF3   R3,*AR0++,R7
    MPYF3   R2,*AR0++,R8
    ADDF    R8,R7
    MPYF3   R1,*AR0++,R8
    ADDF    R8,R7
    MPYF3   R0,*AR0++,R8
    ADDF    R8,R7
    SUBI    3,AR0

MPYF3   R6,*AR1++,R9
    MPYF3   R5,*AR1++,R8
    ADDF    R8,R9
    MPYF3   R4,*AR1++,R8
    ADDF    R8,R9
    SUBF    R9,R7
    STF     R7,*AR1
FILTER_LOOP:
    SUBI    2,AR1


    POP    R9 ; general purpose register
    POP    R8 ; general purpose register
    POP    R7 ; general purpose register
    POP    R6
    POP    R5
    POP    R4
    POP    R3
    POP    R2
    POP    R1
    POP    R0
    POP    AR3
    POP    AR2
    POP    AR1
    POP    AR0
    RETS



/************************************************************************
Function Prototype:
```

```
void vFindFloatPix(int iIndexOfMax, float *pfFftMag, float *pfIndexOfMax)

Author and Date Started: Shah Musa and Kevin Shinpaugh, 2/13/97

Description:
vFindFloatPix(int iIndexOfMax, float *pfFftMag, float *pfIndexOfMax), finds
the index of the peak value of the FFT magnitude of CCD data more precisely
than just an interger.  It uses a Gaussian approximation of the FFT magnitude
of three pixels having the index iIndexOfMax-1, iIndexOfMax, and
iIndexOfMax+1.  The more precise peak value floating point index is stored at
lcoation pointed by pfIndexOfMax.


Input Parameters and Ranges:

int iIndexOfMax:
The integer index having the maximum value of the FFT magnitude.

float *pfFftMag:
pfFfftMag points to (array of 1024) FFT magnitude of the 1024 float data.

float *pfIndexOfMax
The more precise floating point index value which corresponds to the
maximum FFT magnitude.

Returns:
None

Called By:
main ().

*****************************************************************************/

void vFindFloatPix(int iIndexOfMax, float *pfFftMag, float *pfIndexOfMax)
{
      float fLgVal0b1,
            fLgVal2b1,
            fVal0,
            fVal1,
            fVal2,
            fOffset;
      fVal0 = pfFftMag[iIndexOfMax - 1];
      fVal1 = pfFftMag[iIndexOfMax];
      fVal2 = pfFftMag[iIndexOfMax + 1];

      fLgVal0b1 = log(fVal0 / fVal1);
      fLgVal2b1 = log(fVal2 / fVal1);
      fOffset = (fLgVal0b1 - fLgVal2b1)/(2*(fLgVal0b1 + fLgVal2b1));

      *pfIndexOfMax = (float)iIndexOfMax + fOffset;

}



/*****************************************************************************
```

```
Function Prototype:
void vFindTruGap(float fEstGap, float *pfTruGap, float *pfRawData, float
      *pfFourPiByLamda)

Author and Date Started: Shah Musa, 3/15/97

Description:
vFindTruGap( ) finds the more precise gap of the EFPI sensor using the
estimated gap provided.  It does the 4*pi*Gap/lamda transformation (+/- 5
micrometer) around the estimated gap on the 1024 data pointed by pfRawData.
It calls the vDGT( ) function to do the transform, and uses the Golden Search
rule to reduce the number of calls for the vDGT( ) function.  The EFPI gaps
are assumed in micrometer.


Input Parameters and Ranges:

float fEstGap:
fEstGap is the estimated EFPI gap found from the FFT transform.

float *pfTruGap:
pfTruGap points to floating point value where the calculated true gap
be saved.

float *pfRawData:
pfRawData points to the array of 1024 unfiltered CCD data values.

float *pfFourPiByLamda:
pfFourPiByLamda points to the array of 1024 wavelength values which
corresponds
to the 1024 CCD data values pointed to by pfRawData.


Returns:
None

Called By:
main ().

*************************************************************************/

#define           R     0.61803399
#define           C     (1.0 - R)

void vFindTruGap(float fEstGap, float *pfTruGap, float *pfRawData, float
      *pfFourPiByLamda)
{

      float fAx, fBx, fCx;
      float f0 ,f1 ,f2 ,f3 ,fX0 ,fX1 ,fX2, fX3, *fp1, *fp2;
      float tol = 1e-3;

      fAx = fEstGap - 5.0;
      fBx = fEstGap;
      fCx = fEstGap + 5.0;
      fp1 = &f1;
      fp2 = &f2;
```

```
	fX0 = fAx;
	fX3 = fCx;

	if (fabs(fCx-fBx) > fabs(fBx-fAx)){
		fX1 = fBx;
		fX2 = fBx + C * (fCx - fBx);}
	else {
		fX2 = fBx;
		fX1 = fBx - C * (fBx - fAx);
	}
	vDGT( pfRawData, fX1, pfFourPiByLamda, fp1);
	vDGT( pfRawData, fX2, pfFourPiByLamda, fp2);
	while (fabs(fX3-fX0) > tol)
	{
		if (f2 > f1){
			fX0 = fX1;
			fX1 = fX2;
			fX2 = R * fX1 + C * fX3;
			f1 = f2;
			vDGT(pfRawData, fX2, pfFourPiByLamda, fp2);}
		else {
			fX3 = fX2;
			fX2 = fX1;
			fX1 = R * fX2 + C * fX0;
			f2 =f1;
			vDGT(pfRawData, fX1, pfFourPiByLamda, fp1);
		}
	}
	if (f1 > f2){
		*pfTruGap = fX1;}
	else {
		*pfTruGap = fX2;
	}
}



/*************************************************************************
Function Prototype:
void vDGT(float *pfData, float fGap, float *pfFourPiByLamda, float *pfDGTMag)

Author and Date Started: Shah Musa, Kevin Shinpaugh, 2/13/97

Description:
The function, vDGT(float *pfData, float fGap, float *pfFourPiByLamda, float
*pfDGTMag), finds the 4*pi*Gap/lamda transform magnitude,for an EFPI gap value
of fGap, and saves the magnitude in location pointed to by pfDGTMag.

All the lamda values are assumed in nm.

Input Parameters and Ranges:

float *pfData:
pfData is the pointer to the array of 1024 floating point data values for
which the transform magnitude is to be found.
```

```
float fGap:
fGap is the EFPI gap (usually 20.0 to 300.0).

float *pfFourPiByLamda:
pfFourPiByLamda points to the 1024 wavelength values which corresponds to the
data
values pointed by pfData.

float *pfDGTMag:
The location where the Transform magnitude be saved is pointed by pfDGTMag.

Returns:
None

Called By:
vFindTruGap().

*************************************************************************/

#include <math.h>

void vDGT(float *pfData, float fGap, float *pfFourPiByLamda, float *pfDGTMag)
{

      float fParam,
            fReal,
            fImag,
            fSumReal,
            fSumImag;
      int   i;

      fSumReal = 0.0;
      fSumImag = 0.0;
      for (i=0; i<1024; i++){
            fParam = fGap*pfFourPiByLamda[i];
            fReal = pfData[i] * cos(fParam);
            fImag = pfData[i] * sin(fParam);
            fSumReal = fSumReal + fReal;
            fSumImag = fSumImag + fImag;
      }
      *pfDGTMag = (fSumReal*fSumReal + fSumImag*fSumImag);
}



*************************************************************************
* Function Prototype:
* void vDGT(float *pfData, float fGap, float *pfFourPiByLamda,
*           float *pfDGTMag),
* (Implemented in C4x Assembly)
*
* Author and Date Started: Shah Musa, Tracy Cramer, 4/15/97
*
* Description:
* The function, vDGT(float *pfData, float fGap, float *pfFourPiByLamda, float
* *pfDGTMag), finds the 4*pi*Gap/lamda transformation magnitude,for an EFPI
```

```
* gap value of fGap, and saves the magnitude in location pointed to by
* pfDGTMag.
*
* All the lamda values are assumed in nm.
*
*
* Input Parameters and Ranges:
*
* float *pfData: (AR4)
* pfData is the pointer to the array of 1024 floating point data values for
* which the transformation magnitude is to be found.
*
* float *fGap: (AR5)
* fGap is the EFPI gap (usually 20.0 to 300.0).
*
* float *pfFourPiByLamda: (AR6)
* pfFourPiByLamda points to the 1024 wavelength values which corresponds
* to the data values pointed by pfData.
*
* float *pfDGTMag: (AR7)
* The location where the Transform magnitude be saved is pointed by pfDGTMag.
*
* Returns:
* None
*
* Called By:
* vFindTruGap (), (findtrugp.c)
*
*****************************************************************************
.def    _vDGT
_vDGT:
    .file "C:\TI_DSP\TI_TOOLS\MATH.H"
    .globl  _cos
    PUSH    R4  ; fParam
    PUSH    R5  ; fReal & fImag
    PUSH    R6  ; fRealSum
    PUSH    R7  ; fImagSum
    PUSH    R8
    PUSH    AR4 ; pfData
    PUSH    AR5 ; pfGap
    PUSH    AR6 ; pfFourPiByLamda
    PUSH    AR7 ; pfDGTMag

*   Initialize the pointer registers
    LDI     SP,AR0
    LDI     *-AR0(9),AR4 ; pfData
    LDI         *-AR0(10),AR5 ; pfGap
    LDI         *-AR0(11),AR6 ; pfFourPiByLamda
    LDI         *-AR0(12),AR7 ; pfDGTMag

*   Initialize the summing registers to zero
    LDF     0.0,R6 ; (SumReal)
    LDF     0.0,R7 ; (SumImag)

    LDF     *AR5,R8 ;Load Gap in R8

*   (C-statement) for (i=256; i<769; i++)
```

```
*   We are only using 512 array values from the middle of data, so
*   we offset the pointers by 256
    ADDI    256,AR6 ; (pfFourPiByLamda)
    ADDI    256,AR4 ; (pfData)

*   Load the repeat register with (512-1)
    LDI     511,RC
    RPTB    DFT_LOOP
    LDF     *AR6++,R1 ; R1 = 4*pi/lamda

*   (C-statement) fParam =  fGap * pfFourPiByLamda[i];
    MPYF3   R1,R8,R4 ; R4 contains fParam

*   (C-statement) fReal = pfData[i] * cos(fParam);

    PUSH    R4 ; We need to push the parameter for the call to cos
    CALL    _cos ; The return value is in R0 by convention
    SUBI    1,SP ; Decrement stack now that the call is over
    LDF     *AR4++,R4 ; Retrieving fData[i]
    MPYF    R0,R4,R5 ; R5 contains fReal

*   (c-statement) fSumReal = fReal + fSumReal;
    ADDF    R5,R6 ; R6 contains fSumReal

*   (C-statement) fImag = pfData[i] * sin(fParam);
*   Using the identity sin(x) = sqrt(1-cos(x)*cos(x))
    MPYF    R0,R0 ; cos(x)*cos(x)
    SUBRF   1,R0  ; 1-cos(x)^2
    RSQRF   R0,R0 ; 1/sqrt(1-cos(x)^2) (reciprocal square root)
*   Newton-Raphson Iteration
    RCPF    R0,R1
    MPYF3   R1,R0,R2
    SUBRF   2.0,R2
    MPYF    R2,R1 ; End of 1st iteration (16-bit precision)
    MPYF3   R1,R0,R2
    SUBRF   2.0,R0
    MPYF3   R2,R1,R0 ; End of 2nd iteration (32-bit precision). R0 = sin(x)
    MPYF    R0,R4,R5 ; R4 contains pfData[i],R5 contains fImag

DFT_LOOP:
*   (c-statement) fSumImag = fImag + fSumImag;
    ADDF    R5,R7 ; R7 contains fSumImag

*  (C-statement) *pfDGTMag = (fSumReal*fSumReal + fSumImag*fSumImag)
    MPYF    R7,R7,R1 ; R1 contains fSumReal^2
    MPYF    R6,R6,R2 ; R2 contains fSumImag^2
    ADDF    R1,R2 ; R2 contains DGTMag
    STF     R2,*AR7 ; Store the result in memory

    POP     AR7 ; pfDGTMag
    POP     AR6 ; pfFourPiByLamda
    POP     AR5 ; pfGap
    POP     AR4 ; pfData
    POP     R8
    POP     R7  ; fImagSum
    POP     R6  ; fRealSum
    POP     R5  ; fReal & fImag
```

```
    POP     R4  ; fParam
    RETS
```

A.2 Real-Time Signal Processing Routines for the FBG Sensor System

```
/*****************************************************************
Function Prototype:
void main (void)

Author and Date Started: Shah Musa, 5/9/97

Description:
The main() program calls all the initialization, data acquisition, and
signal processing functions.  The functions could either be in C, or
in C4x assembly language.

Method:
The InitC4x() function initializes all the necessary registers of the
TMS320C4x DSP processor.  The InitUART() function initializes all the
necessary registers of the UART for communicating with the comport or
computers.  Ocean Optics CCD spectrometer calibration coefficients are
used to convert any pixel index to wavelenght values.
Every individual CCD spectrometer, having a unique SIE
number has its own conversion coefficients.  The nServUart() serves
the UART request, if any.

The AcqData(int *,DMA3_DEST,DMA4_DEST) acquires the 1100 data set at
the given pointer.  There must be at least 275 (0x44c) memory locations
both at DMA3_DEST and DMA4_DEST.

The rest of the functions are for real-time signal processing.

Any new functions can easily be added in main() at it's proper place,
if necessary.


Input Parameters and Ranges:
None.

Called By:
None.

*****************************************************************/

#include <stdlib.h>
#include <math.h>
```

```
/* If the stack is small, then large arrays must not be kept as local
variables.  Local variables are placed on stack for processing.
The following variables are kept out of main() for
monitoring from the 'watch' window.  */

float
      faData[1100],
      faNomPksWav[2]={850.0,860.0},
      faPks[4],
      faPksWav[2],
      faMicroStrn[2];
short
      nStatus;
int
      iaData[1100],
      iaDMA3_DEST[275],
      iaDMA4_DEST[275],
      iBrgNum=2,
      iaBrgRngs[4]={100,200,300,400},
      iaBrgWdths[4]={8,9,8,9};
float fCoeff1 = 0.24256929,fCoeff2 = -4.176e-5,fIncept = 744.837472;
            /* Calibration Coefficients for the CCD of SIE687 */

void main(void)
{
      int i=0;
      vInitC4x();
      vInitUART();
      for (;;){
            nStatus = nAcqData(iaData,iaDMA3_DEST,iaDMA4_DEST);
                  /* nStatus = 0 , if successful*/
            nStatus = nServUART(iaData);
                  /* sends integer data to UART if requested */
            vFloatData(iaData+16,faData);
                  /* Type conversion, converts data from int to float */
                  /* Actual data starts 16 pixel later */
            vFindPks(faData, iBrgNum, iaBrgRngs, iaBrgWdths,faPks);
            for(i=0;i<2;i++){
                  faPksWav[i] = fIncept + faPks[i]*fCoeff1 +
                        faPks[i]*faPks[i]*fCoeff2;
                  faMicroStrn[i] = 1.2821e6*(faNomPksWav[i]-
                        faPksWav[i])/faNomPksWav[i];
            }

      }
}



*************************************************************************
* Function Prototype:
* vFindPks(float *faData, int iBrgNum, int *iaBrgRngs, int *iaBrgWdths,
* float *faPks)
*
* Author and Date Started: Shah Musa, 8/25/96
*
```

```
* Description:
* The vFindPks() finds out all the Bragg peaks from the 1100 floating point
* data values pointed by faData.
*
*
* Input Parameters and Ranges:
*
* int *faData:
* faData points to the float array where the 1100 acquired data set
* is saved.
*
* int iBrgNum:
* iBrgNum is the total number of Bragg peaks in the 1100 data set.
*
* int *iaBrgRngs:
* iaBrgRngs points to the location where the dynamic ranges of the Bragg
* peaks are stored in terms of pixel indeces.  The ranges must
* be stored in a sequence like: BG1_StPxl, BG1_EndPxl, BG2_StPxl,
* BG2_EndPxl, ...., where BG1 is the leftmost Bragg peak in the 1100
* data set.  For example, the range values can be like: iaBrgRngs={50,
* 190,200,380,450,510,....}.
*
* int *iaBrgWdths:
* iaBrgWdths points to the location where the left and right widths of
* Bragg peaks are stored in terms of number of pixels.  The widhts must
* be stored in a sequence like: BG1_L,BG1_R, BG2_L, BG2_R, ...., where
* BG1 is the leftmost Bragg peak in the 1100 data set.  For example, the
* Bragg width values can be like: iaBrgWdths={8,12,6,9,7,11,....}.
*
*
* Returns:
* None
* Called By:
* main ()
*
*************************************************************************


      .def  _vFindPks

_vFindPks:
      ;Save all the registers being used and modified
      PUSH  RC    ;Save RC
      PUSH  IR0   ;Save IR0
      PUSH  IR1   ;Save IR1
      PUSH  AR0   ;Save AR0
      PUSH  R0    ;Save the lower 32 bits of R0
      PUSHF R0    ;and the upper 32 bits
      PUSH  R1    ;Save the lower 32 bits of R1
      PUSHF R1    ;and the upper 32 bits
      PUSH  R2    ;Save the lower 32 bits of R2
      PUSHF R2    ;and the upper 32 bits
      PUSH  R3    ;Save the lower 32 bits of R3
      PUSHF R3    ;and the upper 32 bits
      PUSH  R4    ;Save the lower 32 bits of R4
      PUSHF R4    ;and the upper 32 bits
      PUSH  R5    ;Save the lower 32 bits of R5
```

```
      PUSHF R5    ;and the upper 32 bits
      PUSH  R6    ;Save the lower 32 bits of R6
      PUSHF R6    ;and the upper 32 bits
      PUSH  R7    ;Save the lower 32 bits of R7
      PUSHF R7    ;and the upper 32 bits
      PUSH  R8    ;Save the lower 32 bits of R8
      PUSHF R8    ;and the upper 32 bits
      PUSH  R9    ;Save the lower 32 bits of R9
      PUSHF R9    ;and the upper 32 bits

      LDI   SP,AR0
      LDI   *-AR0(25),AR6     ;AR6=faData[], (24 PUSHes + 1 = 25)
      LDI   *-AR0(26),R6      ;R6=iBrgNum, (24 PUSHes + 2 = 26)
      LDI   *-AR0(27),AR5     ;AR5=iaBrgRngs[], (24 PUSHes + 3 = 27)
      LDI   *-AR0(28),AR4     ;AR4=iaBrgWdths[], (24 PUSHes + 4 = 28)
      LDI   *-AR0(29),AR3     ;AR3=faPks[], (24 PUSHes + 5 = 29)

      ;(AR6=faData[], is never changed)
BG_LOOP:
      LDI   *AR5++(1),IR0     ;IR0 = BGX_StPxl, (BGx_EndPxl>BGx_StPxl)
      SUBI3 IR0,*AR5++(1),RC  ;RC = BGX_EndPxl - BGX_StPxl
      SUBI  1,RC
      LDF   *+AR6(IR0),R0     ;Init maximum value to the 1st element
      ADDI  1h,IR0            ;Increment IR0

      ;The MAXM loop finds the int pixel # of the Bragg peaks
      RPTB  MAXM
      CMPF  *+AR6(IR0),R0     ;Compare by (R0 - *+AR6(IR0))
      BGE   MAXM
      LDF   *+AR6(IR0),R0     ;If greater, R0 = the new max value (float)
      LDI   IR0,R1            ;R1 = the pixel # having the max value (integer)
MAXM
      ADDI  1h,IR0            ;Increment IR0 by 1
*
*     The following instruction saves the integer pixel # having the max value
*     STF   R1,*AR3++(1)      ;save the pixel # having the max value (integer)
*
*     for the following notaions k_ refers to index and y_ refers to value
      LDI   *AR4++(1),R0      ;R0 = BGX_L
      SUBI3 R0,R1,IR0         ;IR0 = peak pixel # (R1) - BGX_L (R0) = k_left
      LDI   *AR4++(1),R0      ;R0 = BGX_R
      ADDI3 R0,R1,IR1         ;IR1 = peak pixel # (R1) + BGX_L (R0) k_right
      SUBI3 IR0,IR1,RC        ;RC = IR1 - IR0 = Bragg range

      LDF   *+AR6(IR0),R7     ;R7 = y_left
      SUBF3 R7,*+AR6(IR1),R8  ;R8 = y_right - y_left
      FLOAT RC,R0
      RCPF  R0,R7             ;R7 = 1/R0,
      MPYF  R7,R0,R2
      SUBRF 2.0,R2
      MPYF  R2,R7             ;End of 1st iteration (16 bit accuracy)
      MPYF  R7,R0,R2
      SUBRF 2.0,R2
      MPYF  R2,R7             ;End of 2nd iteration (32-bit accuracy)

      MPYF  R7,R8             ;R8 = R8*R7 = slope
```

```
      LDF   0.0,R3            ;R3 holds the sum of the area (=fsum)
      LDF   0.0,R4            ;R4 holds the sum of the moments (=fCentroid)
      LDI   IR0,R2            ;R2 = k_left

      ;The CNTRD loop finds the centroid pixel # in floating point value
      RPTB  CNTRD
      SUBI3 R2,IR0,R9         ;R9=IR0-R2=k-k_left;    IR0=k;
      FLOAT R9,R9
      MPYF  R8,R9             ;R9=R9*R8=slope*(k-k_left)
      ADDF  R7,R9             ;R9=R9+R7=y_corr

      SUBF3 R9,*+AR6(IR0),R5  ;R5=y(k)-y_corr
      ADDF  R5,R3             ;R3=R3+R5   ;fSum=fSum+y(k)-y_corr

      FLOAT IR0,R9            ;convert the pixel index to floating point
      MPYF  R9,R5             ;R5=R5*R2
      ADDF  R5,R4             ;R4=R4+R5   ;fCntrd=fCntrd+(y(k)-y_corr))*k)
CNTRD
      ADDI  1h,IR0            ;Increment IR0 for next repeat

      RCPF  R3,R5             ;R5=1/R3,
      MPYF  R5,R3,R2
      SUBRF 2.0,R2
      MPYF  R2,R5             ; End of 1st iteration (16 bit accuracy)

      MPYF  R5,R3,R2
      SUBRF 2.0,R2
      MPYF  R2,R5             ; End of 2nd iteration (32-bit accuracy)

      MPYF  R5,R4             ;R4=R4*R5(fCentroid=fCentroid/fSum)
      STF   R4,*AR3++(1)      ;save the float peak pixel in IEEE format
      SUBI  1h,R6             ;Decrement R6 (R6 = R6 - 1)
      BP    BG_LOOP           ;Branch if not zero and not negative to BG_LOOP

;Restore all the stacked registers
      POPF  R9    ;Restore the upper 32 bits of R8
      POP   R9    ;and the lower 32 bits
      POPF  R8    ;Restore the upper 32 bits of R8
      POP   R8    ;and the lower 32 bits
      POPF  R7    ;Restore the upper 32 bits of R7
      POP   R7    ;and the lower 32 bits
      POPF  R6    ;Restore the upper 32 bits of R6
      POP   R6    ;and the lower 32 bits
      POPF  R5    ;Restore the upper 32 bits of R5
      POP   R5    ;and the lower 32 bits
      POPF  R4    ;Restore the upper 32 bits of R4
      POP   R4    ;and the lower 32 bits
      POPF  R3    ;Restore the upper 32 bits of R3
      POP   R3    ;and the lower 32 bits
      POPF  R2    ;Restore the upper 32 bits of R2
      POP   R2    ;and the lower 32 bits
      POPF  R1    ;Restore the upper 32 bits of R1
      POP   R1    ;and the lower 32 bits
      POPF  R0    ;Restore the upper 32 bits of R0
      POP   R0    ;and the lower 32 bits
      POP   AR0   ;Restore AR0
      POP   IR1   ;Restore IR1
```

```
      POP   IR0   ;Restore IR0
      POP   RC    ;Restore RC
      RETS
```

A.3 Memory Allocation for the Real-Time Wavelength Modulated Optical Fiber Sensor System

```
/*************************************************************************

Program File:     afss.cmd
Programmer:       Shah Musa
Date:             2/10/97
Description:      This .cmd file declares, in the MEMORY part, all the valid
addresses of the memory spaces available for the C4x system.  In the SECTIONS
part, it instructs the loader where to load the different named or default
sections of the program.

*************************************************************************/


MEMORY
{
      ON_CHIP:    org = 002FF800h,  len = 00000800h
      RS232:      org = 00300000h,  len = 00000008h
      SRAM:       org = 20000000h,  len = 00080000h
      RST_VEC:    org = 80000000h,  len = 00000001h
      NVRAM:      org = 80000100h,  len = 0001E000h
}

SECTIONS
{
      .text       load = SRAM
      .bss:       load = SRAM
      .axl_cos    load = SRAM /* for TARTAN library functions */
      .axl_glb    load = SRAM /* for TARTAN library functions */
      .stack      load = ON_CHIP
}
```

Appendix B

The TCP/IP Client/Server Software

B.1 The TCP/IP Client Graphical User Interface

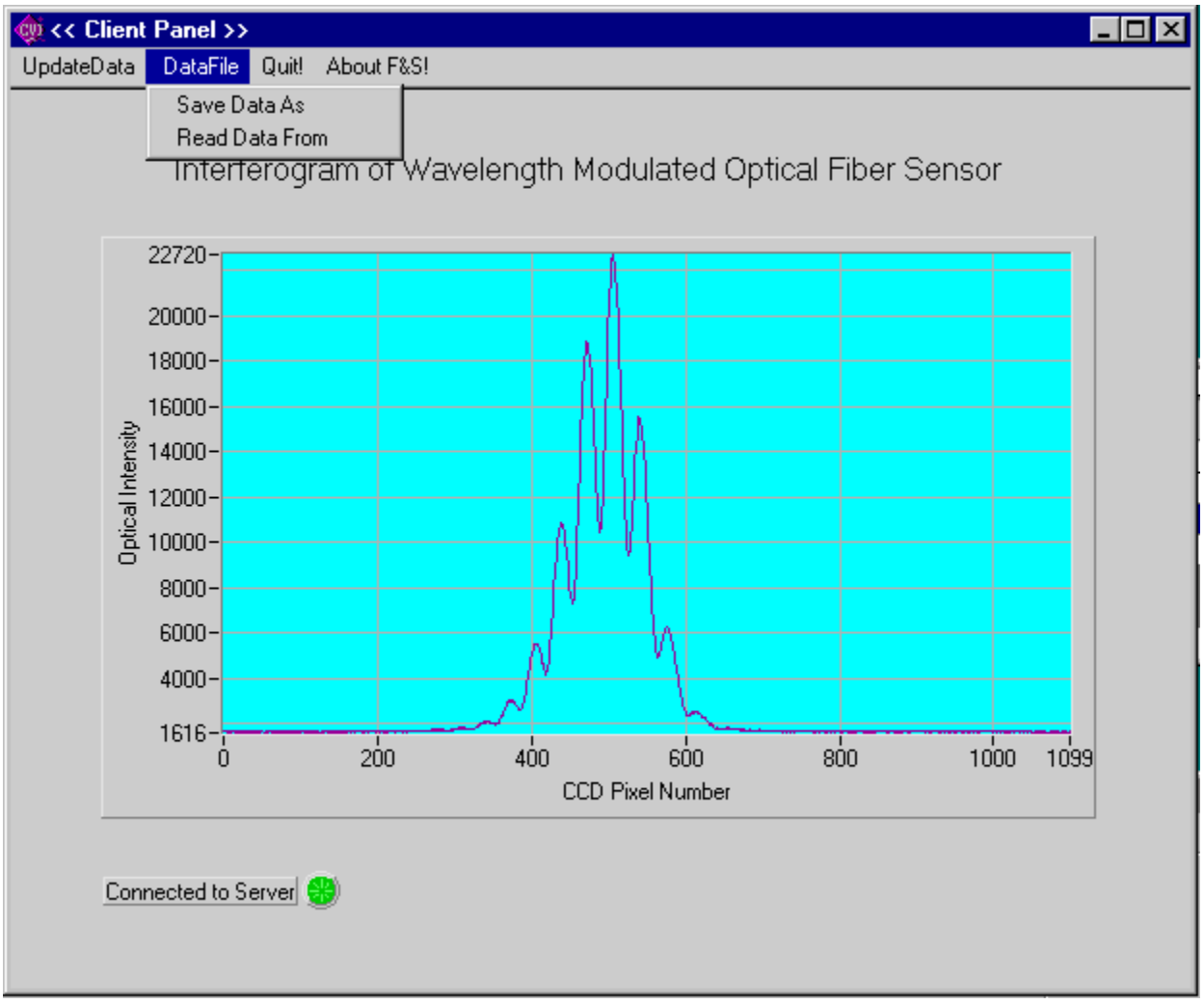

Figure B.1

The TCP/IP client graphical user interface, built with LabWindows-CVI, to acquire real-time data, through the Internet, from the wavelength modulated optical fiber sensor system.

B.2 The TCP/IP Client Software Routines

```
/*
*****************************************************************************
*****************************************************************************

Program File:     client.c
Programmer:       Shah Musa
Date Started:     2/10/97
Purpose:          To run a GUI TCP/IP client. It can request the TCP/IP server
to transmit the 1100 integer CCD data from the 'C40 system through the
Internet either continuously or just one set per request, and dispaly that
data to a graph.  The received data can also be saved in a file, and the saved
data can be read back to the graph at any time.

*****************************************************************************
*/

/* Includes */
#include <tcpsupp.h>
#include <rs232.h>
#include <ansi_c.h>
#include <userint.h>
#include <client.h>
#include <formatio.h>
#include <utility.h>

/* Defines */
#define TRUE 1
#define FALSE 0

/* Global Variables */
short
      bStopUpdate,
      bSingleUpdate,
      nPanelHandle,
      nMenuHandle,
      nStatus;

unsigned
      uConversHandle;

int
      iData[1100];

/* Function Prototypes */
int CVICALLBACK ClientCB(unsigned handle, int event, int error, void
*callbackData);
void vUIErrorMessage(char *, char *, long);


void main(void)
```

```
{
      int
            iPortNum;
      unsigned
            uHandle;
      char
            szServerName[256],
            szPortNum[32];
            *szCallingFunction = "main( )";

      /* prompt for name or IP address of the machine running the server */
      PromptPopup ("Server Name or IP Adress?",
      "Type the name or the IP address of the machine running the server
            \n(example: abc.xyz.com, or, 192.9.202.51)", szServerName, 255);

      /* prompt for the TCP/IP port number of the server to connect to */
      PromptPopup ("Port Number?",
      "Type the TCP/IP port number set for the server program\n(example:
            6000)", szPortNum, 31);
      iPortNum = atoi(szPortNum);

      /* load user interface panel */
      nPanelHandle = LoadPanel (0, "client.uir", PANEL);
      if (nPanelHandle < 0) {vUIErrorMessage(szCallingFunction, "LoadPanel(
            )", nPanelHandle);return;}
      nMenuHandle = GetPanelMenuBar (nPanelHandle);
      if (nMenuHandle < 0) {vUIErrorMessage(szCallingFunction,
            "GetPanelMenuBar( )", nMenuHandle);return;}
      nStatus = DisplayPanel (nPanelHandle);
      if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "DisplayPanel( )",
            nStatus);return;}

      /* Connect to Server */
      if (ConnectToTCPServer (&uHandle, iPortNum, szServerName, ClientCB,
            NULL, 2000)<0) {
            MessagePopup("TCP Client", "Connection to server failed !");
            QuitUserInterface(0);
      } else {
            SetCtrlVal(nPanelHandle, PANEL_CONNECTED, 1);}
      nStatus = RunUserInterface ();
      if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "RunUserInterface(
            )", nStatus);return;}
}


/* Menu callback function */
void CVICALLBACK UpdateCB(int menubar, int menuItem, void *callbackData, int
      panel)
{
      bStopUpdate = FALSE;
      switch (menuItem) {
            case MENU_UPDATEDATA_CONTINUOUS:
                  bSingleUpdate = FALSE;
                  break;
            case MENU_UPDATEDATA_SINGLE:
                  bSingleUpdate = TRUE;
                  break;
```

```
	}
	SetMenuBarAttribute (nMenuHandle, MENU_FILE_SAVEAS, ATTR_DIMMED, 0);
	nStatus = ClientTCPWrite (uConversHandle, "send", sizeof("send"),1000);
	if (nStatus <0) {
		MessagePopup ("TCP Server", "TCP write error!");
		QuitUserInterface(0);
	}
}


/* Menu callback function */
void CVICALLBACK StopUpdateCB(int menubar, int menuItem, void *callbackData,
	int panel)
{
	bStopUpdate = TRUE;
	bSingleUpdate = TRUE;
}


/* Menu callback function */
void CVICALLBACK AboutFNSCB(int menubar, int menuItem, void *callbackData, int
	panel)
{
	MessagePopup("About F&S Inc.","F&S Inc.\ne-mail :: tech_support@f-
		s.com\nHome Page :: www.f-s.com\n");
}


/* Menu callback function */
void CVICALLBACK FileCB(int menubar, int menuItem, void *callbackData, int
	panel)
{
	char  szFileName[260];
	FILE  *fpFile;

	nStatus = FileSelectPopup("","*.dat","*.dat","Download Data
		File",VAL_OK_BUTTON,0,0,1,1,szFileName);
	if (nStatus == 0){
		return; /* The user cancelled the operation */
	}
	if (nStatus < 0){
		vUIErrorMessage("FileCB( )","FileSelectPopup( )",nStatus);
		return;
	}

	switch (menuItem) {
		case MENU_FILE_SAVEAS:
			fpFile = fopen(szFileName,"wb");
				/* Create bianry file to write */
			nStatus = fwrite (iData, sizeof(int), 1100, fpFile);
			if (nStatus != 1100){
				vUIErrorMessage("FileCB( )","fwrite",nStatus);
			}
			fclose(fpFile);
			break;
		case MENU_FILE_READFROM:
			fpFile = fopen (szFileName, "rb");
```

```
                        /* Open bianry file to read */
                    nStatus = fread (iData, sizeof(int), 1100, fpFile);
                    if (nStatus != 1100){
                        vUIErrorMessage("FileCB( )","fread",nStatus);
                    }
                    fclose(fpFile);
                    nStatus = DeleteGraphPlot (nPanelHandle, PANEL_GRAPH, -1,
                        VAL_IMMEDIATE_DRAW);
                    if (nStatus < 0) {vUIErrorMessage("FileCB( )",
                        "DeleteGraphPlot( )",nStatus);return;}
                    nStatus = PlotY (nPanelHandle, PANEL_GRAPH, iData, 1100,
                        VAL_INTEGER, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
                        VAL_SOLID, 1, VAL_DK_MAGENTA);
                    if (nStatus < 0) {
                        vUIErrorMessage("FileCB( )", "PlotY( )", nStatus);
                    }
                    break;
        }
}


/* Menu callback function */
void CVICALLBACK QuitCB(int menubar, int menuItem, void *callbackData, int
        panel)
{
        DisconnectFromTCPServer(uConversHandle);
        QuitUserInterface(0);
}


/* Callback function for the client */
int CVICALLBACK ClientCB(unsigned handle, int event, int error, void
        *callbackData)
{

        int
                iDataSize=1100*sizeof(int),
                i;

        uConversHandle = handle;
        switch (event) {
                case TCP_DATAREADY:
                    if((iDataSize = ClientTCPRead(uConversHandle, iData,
                        iDataSize, 1000))<0) {
                        MessagePopup("TCP Client", "TCP read error!");
                        SetCtrlVal(nPanelHandle, PANEL_CONNECTED, 0);
                        break;
                    }
                    nStatus = DeleteGraphPlot (nPanelHandle, PANEL_GRAPH, -1,
                        VAL_IMMEDIATE_DRAW);
                    if (nStatus < 0) {vUIErrorMessage("ClientCB( )",
                        "DeleteGraphPlot()",nStatus);return 0;}
                    nStatus = PlotY (nPanelHandle, PANEL_GRAPH, iData, 1100,
                        VAL_INTEGER, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
                        VAL_SOLID, 1, VAL_DK_MAGENTA);
                    if (nStatus < 0) {vUIErrorMessage("ClientCB( )", "PlotY( )",
                        nStatus);return 0;}
```

```
                  if (bSingleUpdate != TRUE){
                        nStatus = ClientTCPWrite (uConversHandle, "send",
                              sizeof("send"),1000);
                        if (nStatus <0) {
                              MessagePopup ("TCP Server", "TCP write error!");
                              QuitUserInterface(0);
                        }
                  }
                  break;
            case TCP_DISCONNECT:
                  MessagePopup("TCP Client", "Server has closed connection!");
                  SetCtrlVal(nPanelHandle, PANEL_CONNECTED, 0);
                  break;
      }
      return 0;
}


/*
Error message funtion vUIErrorMessage() pops up the error meassage transferred
to it.  (Modifed from Tracy Cramer's)
*/
void vUIErrorMessage(char *szCallingFunction, char *szCalledFunction, long
      iStatus)
{
      char
            sMessage[128],
            sTemp[128];
      strcpy(sMessage, "Function ");
      strcat(sMessage, szCalledFunction);
      strcat(sMessage, "\ncalled by ");
      strcat(sMessage, szCallingFunction);
      strcat(sMessage, "\nreturned value ");
      Fmt(sTemp, "%s<%i",iStatus);
      strcat(sMessage, sTemp);
      strcat(sMessage, "\n");
      MessagePopup("Error Message", sMessage);
}



/*
*****************************************************************************
*****************************************************************************

Program File:     client.h (Generated by LabWindows-CVI)

*****************************************************************************
* LabWindows/CVI User Interface Resource (UIR) Include File                 *
* Copyright (c) National Instruments 1997. All Rights Reserved.             *
*                                                                           *
* WARNING: Do not add to, delete from, or otherwise modify the contents     *
*          of this include file.                                            *
*****************************************************************************
*/
#include <userint.h>
```

```
#ifdef __cplusplus
    extern "C" {
#endif

     /* Panels and Controls: */

#define  PANEL                          1
#define  PANEL_CONNECTED                2
#define  PANEL_GRAPH                    3


     /* Menu Bars, Menus, and Menu Items: */

#define  MENU                           1
#define  MENU_UPDATEDATA                2
#define  MENU_UPDATEDATA_CONTINUOUS     3 /* callback function: UpdateCB */
#define  MENU_UPDATEDATA_SINGLE         4 /* callback function: UpdateCB */
#define  MENU_UPDATEDATA_STOPUPDATE     5 /* callback function: StopUpdateCB */
#define  MENU_FILE                      6 /* callback function: FileCB */
#define  MENU_FILE_SAVEAS               7 /* callback function: FileCB */
#define  MENU_FILE_READFROM             8 /* callback function: FileCB */
#define  MENU_QUIT                      9 /* callback function: QuitCB */
#define  MENU_ABOUTFNS                 10 /* callback function: AboutFNSCB */


     /* Callback Prototypes: */

void CVICALLBACK AboutFNSCB(int menubar, int menuItem, void *callbackData, int
      panel);
void CVICALLBACK FileCB(int menubar, int menuItem, void *callbackData, int
      panel);
void CVICALLBACK QuitCB(int menubar, int menuItem, void *callbackData, int
      panel);
void CVICALLBACK StopUpdateCB(int menubar, int menuItem, void *callbackData,
      int panel);
void CVICALLBACK UpdateCB(int menubar, int menuItem, void *callbackData, int
      panel);


#ifdef __cplusplus
    }
#endif
```

B.3 The TCP/IP Server Graphical User Interface

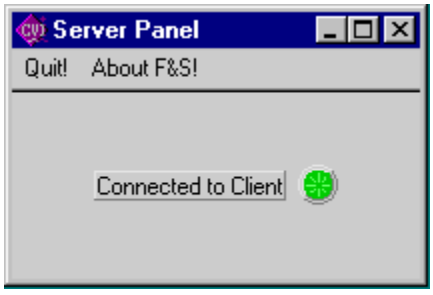

Figure B.2

The TCP/IP server graphical user interface, built with LabWindows-CVI, to transfer real-time data through the Internet, from the wavelength modulated optical fiber sensor system.

B.4 The TCP/IP Server Software Routines

```
/*
*****************************************************************************
*****************************************************************************

Program File:     server.c
Programmer:       Shah Musa
Date Started:     2/10/97
Purpose:          To run a TCP/IP server.  It waits for any client to make
connection to it and request data.  If requested, it downloads the set of 1100
integer CCD data from the 'C40 system through the RS232 comport and transmits
them to the client.

*****************************************************************************
*/

/* Includes */
#include <rs232.h>
#include <ansi_c.h>
#include <userint.h>
#include <formatio.h>
```

```
#include <utility.h>
#include "server.h"
#include <tcpsupp.h>

/* Defines */
#define RS232_BAUD_RATE 38400
#define QBUFFSIZE 512
#define TRUE 1
#define FALSE 0

/* Global Variables */
short
      bStopUpdate,
      bSingleUpdate;
      nPanelHandle,
      nMenuHandle,
      nStatus,
      nComPort=1;
char
      sTemp[260],
      *szComPort = "COM1";
int
      iTCPPort,
      iData[1100];
unsigned
      uHandle;

/* Function Prototypes */
void vRS232Init(void);
void vUIErrorMessage(char *, char *, long);
void vDownloadData(void);
int CVICALLBACK ServerCallback(unsigned handle, int event, int error, void
      *callbackData);


void main(void)
{

      char
      szPortNumString[32],
      *szCallingFunction = "main()";

      /* prompt for port number to connect to the sensor system */
      PromptPopup ("RS232 Communication Port Number?", "Type the com port
            number to connect to the ‘C40 sensor system\n(example: 1 or 2)",
            szPortNumString, 31);
      nComPort = atoi(szPortNumString);
      if (nComPort == 1){
            szComPort = "COM1";}
      else if(nComPort == 2){
            szComPort = "COM2";}
      else {MessagePopup("Communication Port Error", "The Communication Port
            number must be 1 or 2"); return;}
      vRS232Init();

      /* prompt for port number to wait for TCP/IP connection */
```

```
	PromptPopup ("TCP/IP Port Number?", "Type the TCP/IP port number for
		client to connect to\n(example: 6000)", szPortNumString, 31);
	iTCPPort = atoi(szPortNumString);

	nPanelHandle = LoadPanel (0, "server.uir", PANEL);
	if (nPanelHandle < 0) {vUIErrorMessage(szCallingFunction, "LoadPanel()",
		nPanelHandle);return;}
	nMenuHandle = GetPanelMenuBar (nPanelHandle);
	if (nMenuHandle < 0) {vUIErrorMessage(szCallingFunction,
		"GetPanelMenuBar()", nPanelHandle);return;}
	nStatus = DisplayPanel (nPanelHandle);
	if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "DisplayPanel()",
		nPanelHandle);return;}
	if (RegisterTCPServer(iTCPPort, ServerCallback, NULL)<0) {
		MessagePopup ("TCP Server", "RegisterTCPServer failed !!!");
	}

	nStatus = RunUserInterface ();
	if (nStatus < 0) {vUIErrorMessage(szCallingFunction,
		"RunUserInterface()", nPanelHandle);return;}
}


/*
Function vRS232Init() initializes all the necessary parameters of the given
comport
*/
void vRS232Init(void)
{
	char
		*szCallingFunction = "RS232Init()";
	short
		nStatus = 0;
	long
	lBaudRate = RS232_BAUD_RATE;

	nStatus = OpenComConfig(nComPort, szComPort, lBaudRate, 0, 8, 1,
		QBUFFSIZE, QBUFFSIZE);
	if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "OpenComConfig",
		nStatus);return;}
	nStatus = SetComTime (nComPort, 2.0);
	if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "SetComTime",
		nStatus);return;}
	nStatus = SetCTSMode (nComPort, LWRS_HWHANDSHAKE_CTS_RTS);
		/* Set up hardware handshaking */
	if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "SetCTSMode",
		nStatus);return;}
	nStatus = FlushInQ(nComPort);
	if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "FlushInQ",
		nStatus);return;}
	nStatus = FlushOutQ(nComPort);
	if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "FlushOutQ",
		nStatus);return;}
}


/* Menu callback function */
```

```
void CVICALLBACK AboutFNSCB(int menubar, int menuItem, void *callbackData, int
      panel)
{
      MessagePopup("About F&S Inc.","F&S Inc.\ne-mail :: tech_support@f-
            s.com\nHome Page :: www.f-s.com\n");
}


/* Menu callback function */
void CVICALLBACK QuitCB(int menubar, int menuItem, void *callbackData, int
panel)
{
      CloseCom (nComPort);
      UnregisterTCPServer(iTCPPort);
      QuitUserInterface(0);
}


/* Callback function for the server */
int CVICALLBACK ServerCallback(unsigned handle, int event, int error, void
      *callbackData)
{
      char
            caBuf[512];
      int
            iReadSize;

      switch (event) {
            case TCP_CONNECT:
                  uHandle = handle;
                  SetCtrlVal(nPanelHandle, PANEL_CONNECTED, 1);
                  break;
            case TCP_DATAREADY:
                  if((iReadSize = ServerTCPRead(uHandle, caBuf, 512, 1000))==-
                        kTCP_ConnectionClosed) {
                        MessagePopup("TCP Server", "TCP read error!");
                        SetCtrlVal(nPanelHandle, PANEL_CONNECTED, 0);
                        break;
                  }
                  vDownloadData();
                  if(ServerTCPWrite(uHandle, iData, 1100*sizeof(int), 2000)<0)
                  {
                        MessagePopup ("TCP Server", "TCP write error!");
                        SetCtrlVal(nPanelHandle, PANEL_CONNECTED, 0);
                  }
                  break;
            case TCP_DISCONNECT:
                  SetCtrlVal(nPanelHandle, PANEL_CONNECTED, 0);
                  break;
      }
      return 0;
}


/*
The function vDownloadData( ) downloads the 1100 integer data set into the
array iaData[] from the ‘C40 system. It also makes sure that the data is
```

```
synchronized by checking the preamble(iDataBegin) and the post-signature
(iDataEnd) of the data set.
*/
void vDownloadData(void)
{
      int
            i = 0,
            j = 0,
            iDataBegin = 123456789,
            iDataEnd = 987654321,
            iDnldWord = 0;
      char
            *szCallingFunction = "vDownloadData( )";

      for (; ;){
            nStatus = FlushInQ(nComPort);
            if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "FlushInQ(
                  )", nStatus);return;}
            nStatus = FlushOutQ(nComPort);
            if (nStatus < 0) {vUIErrorMessage(szCallingFunction, "FlushOutQ(
                  )", nStatus);return;}

            if (ComWrtByte (nComPort, 'S') !=1){
                  /* Send the command  'S' (0x53) to the 'C40 to initiate the
                        transfer */
                  MessagePopup("Error", "ComWrtByte() error.  Please try
                        again");
                  return;
            }
            nStatus = ProcessSystemEvents ();
            if (nStatus < 0) {vUIErrorMessage(szCallingFunction,
                  "ProcessSystemEvents( )",nStatus);return;}

            for (i=0;i<4;i++){
                  iDnldWord = iDnldWord << 8;
                  iDnldWord = iDnldWord | ComRdByte(nComPort);
            }
            if (iDataBegin == iDnldWord) break;
      }

      for (j=0; j<1100; j++){
            for (i=0;i<4;i++){
                  iDnldWord = iDnldWord << 8;
                  iDnldWord = iDnldWord | ComRdByte(nComPort);
            }
            iData[j] = iDnldWord;
            nStatus = ProcessSystemEvents ();
            if (nStatus < 0) {vUIErrorMessage(szCallingFunction,
                  "ProcessSystemEvents( )",nStatus);return;}

      }
      for (i=0;i<4;i++){
            iDnldWord = iDnldWord << 8;
            iDnldWord = iDnldWord | ComRdByte(nComPort);
      }
      if (iDnldWord != iDataEnd) { MessagePopup ("vDownload Error", "Data Not
            Synchronized!");}
```

```
}


/*
Error message funtion vUIErrorMessage() pops up the error meassage transferred
to it.  (Modifed from Tracy Cramer's)
*/
void vUIErrorMessage(char *szCallingFunction, char *szCalledFunction, long
      iStatus)
{
      char
            sMessage[128],
            sTemp[128];
      strcpy(sMessage, "Function ");
      strcat(sMessage, szCalledFunction);
      strcat(sMessage, "\ncalled by ");
      strcat(sMessage, szCallingFunction);
      strcat(sMessage, "\nreturned value ");
      Fmt(sTemp, "%s<%i",iStatus);
      strcat(sMessage, sTemp);
      strcat(sMessage, "\n");
      MessagePopup("Error Message", sMessage);
}



/*
*****************************************************************************
*****************************************************************************

Program File:     server.h (Generated by LabWindows-CVI)

*****************************************************************************
* LabWindows/CVI User Interface Resource (UIR) Include File                 *
* Copyright (c) National Instruments 1997. All Rights Reserved.             *
*                                                                           *
* WARNING: Do not add to, delete from, or otherwise modify the contents     *
*          of this include file.                                            *
*****************************************************************************
*/
#include <userint.h>

#ifdef __cplusplus
    extern "C" {
#endif

     /* Panels and Controls: */

#define  PANEL                           1
#define  PANEL_CONNECTED                 2


     /* Menu Bars, Menus, and Menu Items: */

#define  MENU                            1
```

```
#define  MENU_QUIT                      2  /* callback function: QuitCB */
#define  MENU_ABOUTFNS                  3  /* callback function: AboutFNSCB */


     /* Callback Prototypes: */

void CVICALLBACK AboutFNSCB(int menubar, int menuItem, void *callbackData, int
      panel);
void CVICALLBACK QuitCB(int menubar, int menuItem, void *callbackData, int
      panel);


#ifdef __cplusplus
    }
#endif
```

Appendix C

The Matlab Codes for the Simulation and the Signal Processing

C.1 Matlab Code to Simulate the Signal Response of the Wavelength Modulated EFPI Sensor System

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program File:   efpi.m
% Programmer:     Shah Musa
% Date:           2/20/97
% Purpose:        To simulate the signal response of the wavelength
% modulated EFPI sensor system, modeled by Equation (2.10).
% The responsivity of the CCD diode elements are assumed independent
% of wavelengths.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
lam   = 745:(960-745)/1099:960;
            % range of the SLED optical source in nm, and the CCD array length
pk    = 500;      % peak amplitude value of the Gaussian SLED source
sigma = 12;       % spread (std. deviation) of the Gaussian SLED source
mu    = 858;      % mean of the source in nm
fs    = pk*exp((-(lam-mu).^2)/(2*sigma^2)); % amplitude value of the source

rpr   = 0.2; % Fresnel's ampl. reflection coeff. at the reference interface
tpr   = 1.2; % Fresnel's ampl. transmission coeff. at the reference interface
rps   = 0.3; % Fresnel's amplitude reflection coeff. at the sensing interface
phi1  = 0;        % reflection phase shift at the reference interface
phi2  = pi;       % reflection phase shift at the sensing interface
a     = 4.0;      % fiber core radius in um
NA    = 0.1;      % fiber numerical aperture
g     = 50 ;      % gap in um

fa_g  = rps*((a/(a+g*tan(asin(NA))))^2);
delta = ( ((4*pi*g*1e-6)./(lam*1e-9)) -phi2 + phi1 );
[x,y] = pol2cart(delta,tpr*fa_g);
ampl  = x + j*y + rpr;
v     = (fs.^2) .* abs(ampl).^2;
v     = v + 50*randn(size(v));

plot(lam,v);
grid;
```

C.2 Matlab Code for the FFT Method of the Wavelength Modulated EFPI Sensor System, to Find the EFPI Gap Length *g*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program File:   rfft.m
% Programmer:     Shah Musa, Kevin Sinpaugh
% Date:           4/15/97
% Purpose:        To find the EFPI gap length g, for an acquired set of
% data, using the FFT method.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

% Curve fitting coefficients for the index of the maximum FFT
% magnitudes, for the CCD spectrometer of SIE687.
fCurvFtCoeff0 = -2.354;
fCurvFtCoeff1 = 1.8196;
fCurvFtCoeff2 = -5.0963e-6;

load c:\mystuff\phdstuff\data\sie687b\g80\g80a.dat;
x = g80a(17:1040); % The 1st active element starts at index 17

[b,a] = butter(3,0.03,'high');
y     = filter(b,a,x);
fFFTMag = abs(fft(y));

fFFTMagMax  = max(fFFTMag);
iIndexOfMaxA = find(fFFTMag == fFFTMagMax); % There are two maximum values
iIndexOfMax = iIndexOfMaxA(1) % The 1st index is used

fNum = log(fFFTMag(iIndexOfMax-1)/fFFTMag(iIndexOfMax)) ...
      - log(fFFTMag(iIndexOfMax+1)/fFFTMag(iIndexOfMax));

fDen = 2*log(fFFTMag(iIndexOfMax-1)/fFFTMag(iIndexOfMax)) ...
      + 2*log(fFFTMag(iIndexOfMax+1)/fFFTMag(iIndexOfMax));

fIndexOfMax = iIndexOfMax + fNum/fDen % Gaussian interpolation

g = fCurvFtCoeff0 + fCurvFtCoeff1*fIndexOfMax ...
      + fCurvFtCoeff2*fIndexOfMax*fIndexOfMax

plot(fFFTMag);
grid;
```

C.3 Matlab Code for the Discrete Gap Transformation on the Data of the Wavelength Modulated EFPI Sensor System, to Find the EFPI Gap Length *g*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program File:   dgt.m
% Programmer:     Shah Musa, Kevin Sinpaugh
% Date:           4/15/97
% Purpose:        To find the EFPI gap length g, for an acquired set of
% data, using the Discrete Gap Transform (DGT) method.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

% Calibration coefficients to convert pixel indices to wavelenghts
% for the CCD spectrometer of SIE687.  (provided by Ocean Optics Inc.)
fCoeff0 = 744.837472;
fCoeff1 = 0.24256929;
fCoeff2 = -4.176e-5;

iPixOffset = 16;  % iPixOffset is the index of the first active element
                  % of the 1100 element CCD linear array spectrometer.
i = [1:1024];
iPix = i + iPixOffset;
fLam = fCoeff0 + fCoeff1*iPix + fCoeff2*(iPix.*iPix); %fLam in nm

load c:\mystuff\phdstuff\data\sie687b\g50\g50a.dat;
fData = g50a(17:1040);  % Only the data of the active elements are used
fDataMax = max(fData);
fData = fData/fDataMax;

n     = 0;
for fGap = 30:0.005:80, % DGT transformation is done in this loop
      fParam = ((4000.0 * pi * fGap) ./ fLam); % fGap in um
      fReal = fData' .* cos(fParam);
      fImag = fData' .* sin(fParam);
      fSumReal = sum(fReal);
      fSumImag = sum(fImag);
      n = n+1;
      fDGTMag(n) = (fSumReal*fSumReal + fSumImag*fSumImag);
            % Square root on fDGTMag(n) is not necessary, as we need only
            % to find the index of the maximum value, not the value itself.
end

fDGTMagMax = max(fDGTMag);
iIndexOfDGTMagMax = find(fDGTMag == fDGTMagMax);

g = 30 + (iIndexOfDGTMagMax-1)*0.005

ga    = [30:0.005:80];
plot(ga,fDGTMag);
grid;
```

References

[Anno92] V. Annovazzi-Lodi, S. Donati, and S. Merlo, "Coiled-fiber sensor for vectorial measurment of magnetic field," *J. Lightwave Technology*, vol. 10, Dec. 1992, pp. 2006-2010.

[Bao95] X. Bao, J. Dhliwayo, N. Heron, D. J. Webb, and D. A. Jackson, "Experimental and theoretical studies on a distributed temperature sensor based on Brillouin scattering," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1340-1348.

[Benn96] K. D. Bennet and C. B. Batroney, "Laser crosshair projection technique for interstory drift measurement," *Proceedings SPIE*, vol. 2839, 1996, pp. 302-310.

[Bert87] J. W. Berthold, W. L. Ghering, and D. Varshneya, "Design and characterization of a high-temperature fiber-optic pressure transducer," *J. Lightwave Technology*, vol. 5, 1987, pp. 1-6.

[Bhat93] V. Bhatia, *Signal Processing Techniques for Optical Fiber Sensors Using White Light Interferometry*, Master's Thesis, Bradley Department of Electrical Engineering, Virginia Tech, 1993.

[Born75] M. Born and E. Wolfe, *Principles of Optics*, Pergamon Press, 5th Ed., 1975.

[Buca77] J. A. Bucaro, H. D. Dardy, and E. F. Carome, "Fiber optic hydrophone," *J. Acoustic Society of America*, vol. 62, 1977, p. 1302.

[Buch95] F. Bucholtz, C. A. Villarruel, A. R. Davis, C. K. Kirkendall, D. M. Dagenais, J. A. McVicker, S. S. Patrick, K. P. Koo, G. Wang, H. Valo, T. Lund, A. G. Andersen, R. Gjessing, E. J. Eidem, and T. Knudsen, "Multichannel fiber-optic magnetometer system for undersea measuremenst," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1385-1395.

[Burr93] Data Sheet, *ADS7810*, Burr-Brown Corporation, 1993, Tucson, AZ.

[Chan84] K. Chan, H. Ito, and H. Inaba, "An optical fiber based gas sensor for remote absorption measurement of low-level CH_4 gas in the near-infrared region," *J. Lightwave Technology*, vol. 2, 1984, pp. 234-237.

[Chen92] S. Chen, A. W. Palmer, K. T. V. Grattan, and B. T. Meggit, "Digital signal-processing techniques for electronically scanned optical-fiber white-light interferometry," *Applied Optics*, vol. 31, Oct. 1992, pp. 6003-6010.

[Clau92] R. O. Claus, M. F. Gunther, A. Wang, and K. A. Murphy, "Extrinsic Fabry-Perot sensor for strain and crack opening displacement measurements from -200 to 900 °C," *Smart Materials and Structures*, vol. 1, 1992, pp. 237-242.

[Code95] User's Guide, *Code Composer: The New Generation of DSP Code Development and Debugging Tools*, GO DSP Corp., Canada, 1995.

[Cole77] J. H. Cole, R. L. Johnson, and P. G. Bhutta, "Fiber optic detection of sound," *J. Acoustic Society of America*, vol. 62, 1977, p. 1136.

[Cork88] M. Corke, F. Gillham, A. Hu, D. W. Stowe, and L. Sawyer, "Fiber optic pressure sensors employing reflective diaphragm techniques," *Proceedings SPIE*, vol. 985, 1988, pp. 164-171.

[Culs95] B. Culshaw, "Introduction to special issue on optical fiber sensors," *J. Lightwave Technology*, vol. 13, July 1995, p. 1187.

[Culs88] B. Culshaw and J. Dakin, *Optical Fiber Sensors: Systems and Applications*, Artech House, MA, 1988.

[Dall95a] Data Sheet, *DS1245Y/AB 1024K Nonvolatile SRAM*, Dallas Semiconductor, 1995.

[Dall95b] Data Sheet, *DS1286 Watchdog Timekeeper*, Dallas Semiconductor, 1995.

[Davi95] M. A. Davis and A. D. Kersey, "Application of a fiber Fourier transform spectrometer to the detection of wavelength-encoded signals from Bragg grating sensor," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1289-1295.

[Debo95] B. J. -C. Deboux, E. Lewis, P. J. Scully, and R. Edwards, "A novel technique for optical fiber pH sensing based on Methylene blue adsorption," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1407-1414.

[DiDo72] M. DiDomenico, Jr., "Material dispersion in optical fiber waveguides," *Applied Optics*, vol. 11, Mar. 1972, pp. 652-654.

[Dils83] R. R. Dils, *J. of Applied Physics*, vol. 84, 1983, p. 1198.

[Druy88] M. A. Druy and W. A. Stevenson, "In situ composite cure monitoring using IR-transmitting optical fibers," *Proceedings SPIE*, vol. 986, 1988, p. 130.

[Fang96] X. Fang et al., "Reciprocity-compensated polarimetric low-birefringence fiber optic current sensor," *J. Lightwave Technology*, vol. 14, July 1996, pp. 1664-1673.

[Fern95] V. Fernicola and L. Crovini, "Digital optical fiber point sensor for high-temperature measurement," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1331-1334.

[Flav95] D. A. Flavin, R. McBride, and J. D. C. Jones, "Interferometric fiber-optic sensing based on the modulation of group delay and first order dispersion: application to strain-temperature measurand," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1314-1323.

[Grat95] K. T. V. Grattan and B. T. Meggit, *Optical Fiber Sensor Technology*, Chapman & Hall, 1995.

[Gree93] P. E. Green, Jr., *Fiber Optic Networks*, Prentice-Hall Inc., New Jersey, 1993.

[Gott81] M. Gottlieb and G. B. Brandt, "Fiber-optic temperature sensor based on internally generated thermal radiation," *Applied Optics*, vol. 20, 1981, p. 3408.

[Habe95] W. R. Habel and H. Polster, "The influence of cementitious building materials on polymeric surfaces of embedded optical fibers for sensors," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1324-1330.

[Hill95] K. O. Hill, B. Malo, K. A. Vineberg, F. Bilodeau, D. C. Johnson, and I Skinner, "Efficient mode conversion in telecommunication fibre using externally written gratings," *Electronics Letters*, vol. 26, August 1990, pp. 1270-1272.

[Hord83] A. Hordrik, A. Berg, and D. Thingbo, "A fiber optic gas detection system," *9th European Conf. Opt. Commun.*, 1983, Geneva, pp. 317-320.

[Hori95] T. Horiguchi, K. Shimizu, T. Kurashima, M. Tateda, and Y. Koyamada, "Development of a distributed sensing technique using Brillouin scattering," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1296-1302.

[Inab79] H. Inaba, T. Kobayashi, M. Hirama, and M. Hamza, "Optical-fibre network system for air pollution monitoring over a wide area by optical absorption method," *Electronics Letters*, Vol. 15, 1979, pp. 749-751.

[Jack96] K. H. Jackel, "Fiber optic displacement measuring system for high-temperature seismic sensor," *Proceedings SPIE,* vol. 2839, 1996, pp. 290-301.

[Jeff85] L. A. Jeffers, "Fiber optic SO_2 analyzer," *Proceedings SPIE*, vol. 566, 1985.

[Jone95] M. E. Jones, *Bragg Grating Sensor Interrogation Using In-Line, Dual-Mode Fiber Demodulator*, Master's Thesis, Bradley Department of Electrical Engineering, Virginia Tech, 1995.

[Keis91] G. Keiser, *Optical Fiber Communications*, Second Edition, McGraw-Hill Inc., New York, 1991.

[Kuri83] M. Kuribara, "Liquid-core optical fiber for voltage measurement using the Kerr effect," *Electronics Letters*, vol. 19, 1983, pp. 133-135.

[Laga81] N. Lagakos, T. Litovitz, P. Macedo, and R. Mohr, "Multimode optical fiber displacement sensor," *Applied Optics*, vol. 20, 1981, pp. 167-168.

[Laws83] C. M. Lawson and V. J. Tekippe, " Environmentally insensitive diaphragm reflectance pressure sensor," *Proceedings SPIE*, vol. 412, 1983, pp. 96-103.

[Lesk92] J. J. Lesko, G. P. Carman, B. R. Fogg, W. V. Miller, A. M. Vengsarkar, K. L. Reifsnider, and R. O. Claus, "Embedded Fabry-Perot fiber optic strain sensors in the macromodel composites," *Optical Engineering*, vol. 31, Jan. 1992, pp. 13-22.

[Lima96] N. I. Limanova, "Multichannel fiber optic sensors for precision measurements of vibration and linear position," *Proceedings SPIE*, vol. 2839, 1996, pp. 342-349.

[Luo96] F. Luo, A Meng, J. Liu, K. Kim, and D. Park, "Optical fiber sensing method for the in-situ cure monitoring of composite materials," *Proceedings SPIE*, vol. 2839, 1996, pp. 413-419.

[Mali96] A. Malik, *A Dual Wavelength Fiber Optic Strain Sensing System*, Master's Thesis, Bradley Department of Electrical Engineering, Virginia Tech, 1996.

[Mans84] S. Mansouri and J. S. Schultz, " A miniature optical glucose sensor based on affinity binding," *Bio Technology*, 1984, pp 885-889.

[Mark81] D. R. Markle, D. A. McGuire, S. R. Goldstein, R. E. Patterson, and R. M. Watson, "A pH measurement system for use in tissue and blood employing miniature fiber optic probes," *Advances in Bioengineering*, Ed: D. C. Viano (Am. Soc. Mech. Eng., New York), p. 123.

[Mars96] R. H. Marshall et al., "Mach-Zehnder electronically scanned white-light interferometer," *J. Lightwave Technology*, vol. 14, March 1996, pp. 397-402.

[Meas92] R. M. Measures, "Advances toward fiber optic based smart structures," *Optical Engineering*, vol. 31, Jan. 1992, pp. 34-47.

[Melt96] G. Meltz, "Overview of fiber grating-based sensors," Proceedings SPIE, vol. 2838, 1996, pp. 2-22.

[Meun95] C. Meunier, J. J. Guerin, M. Lequime, M. Rioual, E. Noel, D. Eguiazabal, D. Fleury, J. Maurin, and R. Mongin, "Industrial prototype of a fiber-optic sensor network for the thermal monitoring of the turbogenerator of a nuclear power plant -design, qualification, and settlement," *J. Lightwave Technology*, vol. 13 , July 1995, pp. 1354-1361.

[Mich95] W. C. Michie, B. Culshaw, M. Konstantaki, I. McKenzie, S. Kelly, N. B. Graham, and C. Moran, "Distributed pH and water detection using fiber optic sensors and hydrogels," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1415-1420.

[Mign95] A. G. Mignani and F. Baldini, "In-vivo biomedical monitoring by fiber-optic system," *J. Lightwave Technology*, vol. 13, July 1995, pp. 1396-1406.

[Mila83] F. P. Milanovich and T. Hirschfeld, "Process, product, and waste stream monitoring with fibre optics," *Advances in Instrumentation*, *Proceedings ISA Int. Conf*. vol. 38, 1983, pp. 407-418.

[Moll88] K. D. Moller, *Optics*, University Science Books, California, 1988.

[Murp91] K. A. Murphy, M. F. Gunther, A. M. Vengsarkar, and R. O. Claus, "Quadrature phase-shifted, extrinsic Fabry-Perot optical fiber sensors," *Optics Letters*, vol. 16, Feb. 1991, pp. 273-275.

[Murp92a] K. A. Murphy, *Novel Phase-Modulated Optical Fiber Sensors*, Ph. D. Dissertation, Bradley Department of Electrical Engineering, Virginia Tech, 1992.

[Murp92b] K. A. Murphy, B. R. Fogg, and A. M. Vengsarkar, "Spatially weighted vibration sensors using tapered two-mode optical fibers," *J. Lightwave Technology*, vol. 10, Nov. 1992, pp. 1680-1687.

[Nati95] Data Sheet, *PC16550D Universal Asynchronous Receiver/Transmitter with FIFOs*, National Semiconductor Corp., 1995.

[Nell96] P. M. Nellen, R. Bronnimann, U. J. Sennhauser, C. G. Askins, and M. A. Putnam, "Strain measurement on concrete beam and carbon fiber cable with distributed optical fiber Bragg grating sensors," *Optical Engineering*, vol. 35, Dec. 1996, pp. 2570-2577.

[Ocea93] *OEM Interface Guide, S1000 Linear Array Spectrometer*, Ocean Optics, Inc., Dunedin FL, Jan. 1993.

[Pete84] I. J. Peterson and C. G. Vurek, "Fiber optic sensors for biomedical applications," *Science*, 1984, vol. 224, pp. 123-127.

[Pete80] I. J. Peterson, S. R. Goldstein, R. V. Fitzgerald, and D. K. Buckhold, "Fibre optic pH probe for physiological use," *Anal. Chem*., vol. 52, 1980, p. 864.

[Pres88] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes in C, the Art of Scientific Computing*, Cambridge University Press, 1988.

[Rama88] S. Ramakrishnan, L. Unger, and R. Kist, "Line loss independent fiber optic displacement sensor with electrical subcarrier phase encoding, *5th Int. Conf. Optical Fiber Sensors*, OFS '88, New Orleans, pp. 133-136.

[Roge73] A. J. Roge, "Optical technique for measurement of current at high voltage," *Proceedings IEE*, vol. 120, 1973, pp. 261-267.

[Sigt93] Reference Manual, *SigTar, Optimized Signal Processing Library for C programmers on the TMS320C40 DSP*, ver. 1.0, Tartan Inc., 1993.

[Shin92] K. A. Shinpaugh, R. L. Simpson, A. L. Wicks, S. M. Ha, and J. L. Fleming, "Singal-processing techniques for low signal-to-noise ratio laser Doppler velocimetry signals," *Experiments in Fluids*, vol. 12, 1992, pp. 319-328.

[Shin97] K. Shinpaugh, F&S Inc., personal conversation, Jan., 1997

[Sirk95] J. Sirkis, T. A. Berkoff, R. T. Jones, H. Singh, A. D. Kersey, E. J. Friebele, and M. A. Putnam, "In-line fiber etalon (ILFE) fiber-optic strain sensors, *J. Lightwave Technology*, vol. 13, July 1995, pp. 1256-1263.

[Stat95] *Static RAMs Data Book*, NEC Electronics Inc., 1995.

[Tana75] T. Tanaka and G. B. Benedeck, "Measurement of the velocity of blood flow (in vivo) using a fiber optic catherter and optical mixing spectroscopy," *Applied Optics*, vol. 14, 1975, p. 189.

[Tms93] *TMS320C4x User s Guide*, Texas Instruments, 1993.

[Tran91] T. A. Tran, *Extrinsic Fabry-Perot Interferometer for Surface Acoustic Wave Measurement*, Master's Thesis, Bradley Department of Electrical Engineering, Virginia Tech, 1991.

[Udd91] E. Udd, *Fiber Optic Sensors An Introduction for Engineers and Scientists*, John Wiley & Sons Inc., 1991.

[Urru88] E. H. Urruti, P. E. Blaszyk, and R. M. Hawk, "Optical fibers for structural sensing applications," *Proceedings SPIE*, vol. 986, 1988, p. 158.

[Vect93] Reference Manual, *VecTar, High Performance Vector Math Library for C programmers on the TMS320C40 DSP*, ver. 1.0, Tartan Inc., 1993.

[Veng96] A. M. Vengsarkar, P. J. Lemaire, J. B. Judkins, V. Bhatia, T. Erdogan, and J. E. Sipe, "Long-period gratings as band-rejection filters," *J. Lightwave Technology*, vol. 14, Jan. 1996, pp. 58-65.

[Wade85] C. A. Wade, J. P. Daikin, J. Croft, and J. Wright, "Optical fibre displacement sensor based on electrical subcarrier interferometry using a Mach Zehnder configuration," *Proceedings SPIE*, vol. 586, 1985, p. 223.

[Wang95] G. Z. Wang, A. Wang, R. G. May, and R. O. Claus, “Self-referenced fiber optic sensor for microdisplacement measurement,” *Optical Engineering*, vol. 34, Jan. 1995, pp. 240-243.

[Wang92] A. Wang, S. He, X. Fang, X. Jin, and J. Lin, “Optical fiber pressure sensor based on photoelasticity: application,” *J. Lightwave Technology*, vol. 10, Oct. 1992, pp. 1466-1472.

[Wood89] R. L. Wood, A. K. Tay, and D. A. Wilson, “Design and Fabrication considerations for composite structures with embedded fiber optic sensors,” *Proceedings SPIE*, vol. 1170, 1989, p. 160.

[Xilin94] *The Programmable Logic Data Book*, Xilinx Inc., 1994.

[Yu95] F. T. Yu, J. Zhang, K. Pan, D. Zhao, and P. B. Ruffin, “Fiber vibration sensor that uses the speckle contrast ratio,” *Optical Engineering*, vol. 34, Jan. 1995, pp. 236-239.

[Zimm89] B. D. Zimmerman, R. O. Claus, D. Kapp, and K. A. Murphy, “Optical time domain reflectometry for local strain measurements,” *Proceedings SPIE*, vol. 1170, 1989, p534.

Vita

Shah Mohammed Musa was born on August 7, 1966 in Barisal, Bangladesh. He attended classes at Barisal Zilla School, Dhaka College, and Barisal B. M. College. He received the Bachelor of Science degree in Electrical and Electronics Engineering from Bangladesh University of Engineering & Technology (BUET) in 1991, and the Master of Science degree in Electrical Engineering from Virginia Tech in 1994.

He has worked the summer of 1993 with Alcatel, Roanoke, to design and build systems with programmable logic control devices, and the summer of 1994 with FiberCom, Roanoke, to develop software to simulate the interworking of Frame Relay and ATM networks. For the last three summers, 1994 to 1996, he has been working with F&S Inc., Blacksburg, designing chips with Xilinx FPGAs, developing software for data acquisition, and designing and developing TI TMS320C4x DSP processor based real-time optical fiber sensor systems. From Fall 1993 until Spring 1997, for every regular semester, he has served Bradley Department of Electrical Engineering of Virginia Tech, either as a research assistant, designing and developing real-time signal processing systems, or as a teaching assistant, grading courses and teaching labs. Recently he has accepted an offer from Intel Corporation, Santa Clara, to work with the development of a new generation of Intel architecture, a 64-bit processor code named ‘Marced’.

He is a student member of the Institute of Electrical and Electronic Engineers (IEEE).