```
#include <fat.h>
#ifndef NDEBUG
  #include <cross_studio_io.h>
#endif

FAT_DEFINITION_t w2FAT;

FAT_DEFINITION_t *w2FAT_build(void)
{
//-----------------------------------------------------------------------------
// Build the FAT table - fixed structure based on the external flash attached
//-----------------------------------------------------------------------------
  UINT32  sector, entry;

  w2FAT.fat_version = FAT_VERSION;
  w2FAT.blockSize = EVENT_ALLOCATION;
  w2FAT.free_entries = NUMBER_OF_EVENTS;
  w2FAT.sector_to_erase = 0;
  for (sector = 0; sector < NUM_SECTOR; sector++)
  {
    w2FAT.sectorAddr[sector] = SECTOR_BYTES * sector;
    for (entry = 0; entry < EVENTS_PER_SECTOR; entry++)
    {
      w2FAT.fat[sector][entry].addr = w2FAT.sectorAddr[sector] + (entry *
      EVENT_ALLOCATION);
      w2FAT.fat[sector][entry].free = true;
      w2FAT.fat[sector][entry].index = (sector * EVENTS_PER_SECTOR) + entry;
    }
  }
  return &w2FAT;
}

FAT_ENTRY_t *w2FAT_findfree(void)
{
  UINT8   sector, entry;

  if (w2FAT.free_entries == 0)
    return (FAT_ENTRY_t *)NOFLASH;
  for (sector = 0; sector < NUM_SECTOR; sector++)
    for (entry = 0; entry < EVENTS_PER_SECTOR; entry++)
      if (w2FAT.fat[sector][entry].free == true)
      {
        w2FAT.free_entries--;
        return &w2FAT.fat[sector][entry];
      }
  return (FAT_ENTRY_t *)NOFLASH;
}

ERROR_STATE w2FAT_freeall(void)
{
//-----------------------------------------------------------------------------
// Only way to return a sector to available is to erase
//-----------------------------------------------------------------------------
  UINT8   sector, entry;
  BE();
  w2FAT.free_entries = NUMBER_OF_EVENTS;
  for (sector = 0; sector < NUM_SECTOR; sector++)
    for (entry = 0; entry < EVENTS_PER_SECTOR; entry++)
```

```
      w2FAT.fat[sector][entry].free = true;
  return OK;
}

UINT16  w2FAT_freemem(void)
{
  UINT8   sector, entry;
  UINT16  freeentries = 0;
  UINT16  used;

  for (sector = 0; sector < NUM_SECTOR; sector++)
    for (entry = 0; entry < EVENTS_PER_SECTOR; entry++)
      if (w2FAT.fat[sector][entry].free == true) freeentries++;
  used = (UINT16)(freeentries*100/NUMBER_OF_EVENTS);
  return used;
}
```

```
#ifndef __spi
#define __spi

#include <eddef.h>
#include <string.h>

#define TERMINATE_ON_RECEIVE     0
#define TERMINATE_ON_TX_EMPTY    1
#define TERMINATE_ON_TX_READY    2

#define SPIIRQON   IFG2 &= ~URXIFG1; IE2 |= URXIE1
#define SPIIRQOFF IE2 &= ~URXIE1

extern UINT8 *U1RXBuffer;

ERROR_STATE SPI_init(void);
UINT8 SendSPIByte(UINT8 value, UINT8 terminate);
void  SendSPIByteBlock(void *src, UINT16 nBytes, UINT8 terminate); // send
consecutive blocks of data
void  ReadSPIByteBlock(void *dest, UINT16 nBytes); // read consecutive blocks


#endif
```

```
#ifndef __accel
#define __accel

#include <eddef.h>

//---------------------------------------------------------------------------
// This module deals with the TI accelerometers
//---------------------------------------------------------------------------

#define GETXYSTATUS    ((P4IN & BIT3) == 0)
#define GETZSTATUS     ((P4IN & BIT4) == 0)
#define SETXYTEST      SETBITS(P4OUT, BIT1)
#define SETZTEST       SETBITS(P4OUT, BIT2)
#define CLEARXYTEST    CLEARBITS(P4OUT, BIT1)
#define CLEARZTEST     CLEARBITS(P4OUT, BIT2)
#define PORT_AS_PWM    SETBITS(P4SEL, (BIT1 | BIT2))
#define PORT_NORMAL    CLEARBITS(P4SEL, (BIT1 | BIT2))

typedef enum  { INIT, UPDATE_BASE, UPDATE_PWM, CLEAR, CLOSE } PWM_STATE;

ERROR_STATE MMA1201_init(void); // reset the accelerometer and make sure passes
self-test
ERROR_STATE MMA3201_init(void); // reset the accelerometer and make sure passes
self-test
ERROR_STATE MMA1201_self_test(BOOL state); // sets the self-test bit
ERROR_STATE MMA3201_self_test(BOOL state); // sets the self-test bit
ERROR_STATE MMA1201_pwm(PWM_STATE state, UINT16 base, UINT16 on);
ERROR_STATE MMA3201_pwm(PWM_STATE state, UINT16 base, UINT16 on);
ERROR_STATE MMA1201_test(void); // send PWM signal to the sensor to simulate
accident
ERROR_STATE MMA3201_test(void); // send PWM signal to the sensor to simulate
accident

#endif
```

```
#ifndef __ADC
#define __ADC

#include <eddef.h>
#include <msp430x14x.h>

UINT16  getADCsamples(UINT8 []);
INT16 ADCtoG(UINT8 sample, long num, long den, long intercept);
UINT8 GtoADC(INT16 sample, long num, long den, long intercept, UINT8 *zero, int
*offset);

#endif
```

```
/*****************************************************************************
**
*
*     FILE NAME:  eddef.h
*
*     DESCRIPTION:  Useful standard definitions
*
*     AUTHOR:  EMM 01/24/2006
*
*****************************************************************************
*/

#ifndef __eddef
#define __eddef

#include <msp430x14x.h>

typedef signed char                     INT8;
typedef unsigned char                   UINT8;
typedef unsigned int                    UINT16;
typedef unsigned long                   UINT32;
typedef long                            INT32;
typedef enum unsigned char { false = 0x00, true = 0xff } BOOL;
typedef int                             INT16;
typedef enum  { OK, FAIL, ERROR }       ERROR_STATE;
typedef enum  { S8 = 0xFF, S7 = 0xFE, S6 = 0xFC, S5 = 0xF8, S4 = 0xF0, S3 =
0xE0, S2 = 0xC0, S1 = 0x80, S0 = 0x00 } FLASH_STATES; // values that can be
written to flash to indicate a state
typedef enum unsigned char { FREE = 0xFF, NEW = 0xFE, PROCESSED = 0xFC, SENT =
0xF8, LOCKED = 0xF0, BELOW = 0x01, DELETED = 0x00} EVENT_STATE;

#define NULL       0
#define NULLP      ((void *) 0)
#define CR         0x0D
#define LF         0x0A
#define SPACE      0x20
#define ACK        6
#define NAK        21
#define EOT        0x04
#define NOFLASH    0xFFFFFFFF

// Bit operation macros
#define TOGGLEBITS(VAR,MASK)      VAR ^= (MASK)
#define CLEARBITS(VAR,MASK)       VAR &= ~(MASK)
#define SETBITS(VAR,MASK)         VAR |= (MASK)
#define BITSET(VAR,MASK,VALUE)    VAR = (VALUE) ? (VAR | (MASK)) : (VAR &
~(MASK)) // General form

// Math operations
#define ABS32(value)        (((long)(value) < 0) ? -(long)(value) :
(long)(value))
#define ABS16(value)        (((int)(value) < 0) ? -(int)(value) : (int)(value))
#define ABS8(value)         (((INT8)(value) < 0) ? -(INT8)(value) :
(INT8)(value))
#define INRANGE(V,LO,HI)    (((V)>=LO) && ((V)<=HI))
#define CLIPTO(V,HI,LO)     V = ((V) > HI) ? HI : ((V) < LO) ? LO : (V)
#define DEC(A)              if (A > 0) A--
#define INC(A,LIMIT)        if (A < LIMIT) A++
```

```
#define MAKEBOOL(TEST)     ((TEST) ? true : false)
#define ROUNDUP(VAL, BASE)   (((VAL) % BASE) > 0) ? (((VAL)/BASE+1)*BASE) :
(VAL)

#endif
```

```
#ifndef __uart
#define __uart

#include <msp430x14x.h>
#include <eddef.h>
#include <string.h>
#include <stdio.h>
#include <crc.h>
#include <ctype.h>
#include <triggers.h>

#define UART_BUF_LEN       64
#define START_MESSAGE      0
#define CONTINUE_MESSAGE   1
#define CLOSE_MESSAGE      2
#define CHECKSUM_ONLY      3
#define CHECKSUM_NOSEP     4


typedef struct {
  char    OutputBuffer[UART_BUF_LEN];
  char    U0RXBuffer[UART_BUF_LEN];
  UINT16  RXIndex;
  UINT16  TXIndex;
  UINT8   nTXBytes;
  BOOL    Ready;
  UINT16  CRC;
  UINT16  sentBytes;
  UINT8   msgEnd;
  BOOL    ACKreceived;
  BOOL    NAKreceived;
  BOOL    replyReceived;
} UART_t;

#define UART0_RX_IRQON       IE1 |= URXIE0
#define UART0_RX_IRQOFF      IE1 &= ~(URXIE0)
#define UART0_TX_IRQON       IE1 |= UTXIE0 // Disable USART0 interrupts
#define UART0_TX_IRQOFF      IE1 &= ~(UTXIE0)  // Disable USART0 interrupts
#define UART0_FULL           ((IFG1 & UTXIFG1) == 0)
#define UART0_SHIFTING       ((UTCTL0 & TXEPT) == 0)
#define UART0_BUSY           (myUART.Ready == false)
#define CLEAR_ACK_NAK        myUART.replyReceived = myUART.ACKreceived =
myUART.NAKreceived = false

extern UART_t myUART;

void  InitUART0(UINT16 baud);
void  SendEvent(UINT8 id, unsigned long ulPacked);
void  SendBuffer(const void *buffer, const UINT8 length, BOOL newMsg);
ERROR_STATE SendMCMMessage(const void *str, UINT8 length, UINT8 code);
ERROR_STATE SendMCMLine(char *str);
ERROR_STATE SendDebugMessage(const char *str);

#endif
```

```
#include <RTC.h>
//-------------------------------------------------------------------------
// This module communicates with the EPSON 4574 RTC
//-------------------------------------------------------------------------

#define RTC_INIT            P6DIR |= (BIT5 | BIT6 | BIT7)
#define RTC_CLK_HIGH        P6OUT |= BIT6
#define RTC_CLK_LOW         P6OUT &= ~BIT6
#define RTC_WAIT            __delay_cycles(4)
//#define PULSE_CLOCK       RTC_CLK_HIGH; RTC_WAIT; RTC_CLK_LOW; RTC_WAIT
#define PULSE_CLOCK         RTC_CLK_HIGH; RTC_CLK_LOW;
#define RTC_DATA_READ       P6DIR &= ~BIT5
#define RTC_DATA_WRITE      P6DIR |= BIT5
#define RTC_ASSERT_CS       P6OUT |= BIT7
#define RTC_DEASSERT_CS     P6OUT &= ~BIT7
#define RTC_ONE             P6OUT |= BIT5
#define RTC_ZERO            P6OUT &= ~BIT5
#define RTC_GET_BIT         (P6IN & BIT5) ? 1 : 0

#define BINTOBCD(VAL) (((VAL) / 10) << 4) | ((VAL) % 10)
#define BCDTOBIN(VAL) (((VAL) >> 4) * 10) + ((VAL) & 0x0F)

BOOL  rtc_good = false;

ERROR_STATE RTC_send(UINT8 data, UINT8 bits)
{
  UINT16  i;
// Send the command sequence
  RTC_DATA_WRITE;
  for (i=0;i<bits;i++)
  {
    if (data & 0x01) RTC_ONE; else RTC_ZERO;
    data >>= 1;
    PULSE_CLOCK;
  }
}

UINT8 RTC_read(void)
{
  UINT8 value, i;

  RTC_DATA_READ;
  value = 0;
  for (i=0;i<8;i++)
  {
    value |= (RTC_GET_BIT) << i;
    PULSE_CLOCK;
  }
  return value;
}

ERROR_STATE RTC_init(void)
{
//-------------------------------------------------------------------------
// Checks whether the RTC's oscillator has stopped
//-------------------------------------------------------------------------
  UINT8 reply;
  RTC_INIT;
```

```
  RTC_ASSERT_CS;
  PULSE_CLOCK;
  RTC_send(0x0C, 4);
  RTC_send(0x00, 4);
  reply = RTC_read();
  RTC_DEASSERT_CS;
  RTC_WAIT;
  RTC_ASSERT_CS;
  PULSE_CLOCK;
  RTC_send(0x03,4);
  RTC_send(0x0C,4);
  RTC_send(0b10010000,8); // Configure the timer for 64Hz time base
  RTC_send(0b00010000,8); // Configure the time base for 5Hz interrupts
  RTC_send(0b00010001,8); // Enable the timer interrupt
  RTC_DEASSERT_CS;
  rtc_good = (reply & 0x80) ? false : true;
  return (rtc_good) ? OK : FAIL;
}

ERROR_STATE set_RTC_date(const char *datetime)
{
//-----------------------------------------------------------------------------
// checks to see if the oscillator has stopped since the MSP reset
// returns OK if the system never stopped and FAIL if the clock has stopped
//-----------------------------------------------------------------------------
  UINT8   sec, min, hr, day, month;
  UINT16  year;
  UINT8   count;

  if (strcmp(datetime, "00:00:00:00:00:0000") == 0) return FAIL;
// Parse the date/time string
  count = sscanf(datetime, "%d:%d:%d:%d:%d:%d", &sec, &min, &hr, &day, &month,
  &year);
  if (count != 6) return FAIL;
// Convert values into BCD representation
  sec = BINTOBCD(sec);
  min = BINTOBCD(min);
  hr = BINTOBCD(hr);
  day = BINTOBCD(day);
  month = BINTOBCD(month);
  year = BINTOBCD(year - 2000);
// Send data across
  RTC_ASSERT_CS;
  PULSE_CLOCK;
  RTC_send(0x03,4);
  RTC_send(0x0F,4);
  RTC_send(0b00110000,8); // This stops the clock for writing
  RTC_send(sec,8);         // write seconds
  RTC_send(min,8);         // write minutes
  RTC_send(hr,8);        // write hour
  RTC_send(1,8);           // write dow
  RTC_send(day,8);         // write day
  RTC_send(month,8);         // write month
  RTC_send(year,8);        // write year
  RTC_DEASSERT_CS;
  RTC_WAIT;
  RTC_ASSERT_CS;
  PULSE_CLOCK;
```

```
  RTC_send(0x03,4);
  RTC_send(0x0C,4);
  RTC_send(0b10010000,8); // Configure the timer for 64Hz time base
  RTC_send(0b00010000,8); // Configure the time base for 4Hz interrupts
  RTC_send(0b00010001,8); // Enable the timer interrupt
  RTC_send(0b00000000,8); // This restarts the clock
  RTC_DEASSERT_CS;
}

void get_RTC_date(char *str)
{
//-------------------------------------------------------------------------
// checks to see if the oscillator has stopped since the MSP reset
// returns OK if the system never stopped and FAIL if the clock has stopped
// char *datetime should point to a char array with 20 characters -
null-terminated string
// returned as sec:min:hr day/month/year(4)
//-------------------------------------------------------------------------
  UINT8 i;
  UINT8 reply[7];
  UINT16 year;

  RTC_ASSERT_CS;
  PULSE_CLOCK;
  RTC_send(0x0C, 4);
  RTC_send(0x00, 4);
  for (i=0;i<7;i++) reply[i] = RTC_read();
  RTC_DEASSERT_CS;
// Convert BCD values to binary
  reply[0] = BCDTOBIN(reply[0] & 0x7F);
  reply[1] = BCDTOBIN(reply[1] & 0x7F);
  reply[2] = BCDTOBIN(reply[2] & 0x7F);
  reply[4] = BCDTOBIN(reply[4] & 0x7F);
  reply[5] = BCDTOBIN(reply[5] & 0x7F);
  year = BCDTOBIN(reply[6] & 0x7F) + 2000;
  sprintf(str, "%02d:%02d:%02d:%02d:%02d:%04d", reply[0], reply[1], reply[2],
  reply[4], reply[5], year);
}
```

```
/*****************************************************************************
**
*
*     FILE NAME:  eddef.h
*
*     DESCRIPTION:  Useful standard definitions
*
*     AUTHOR:  EMM 01/24/2006
*
*****************************************************************************
*/

#ifndef __lists
#define __lists

#include <eddef.h>

typedef INT16 LISTDATA_t; // This is list data type
typedef struct clist
{
  LISTDATA_t   *begin;
  LISTDATA_t   *end;
  LISTDATA_t   *last;
  UINT16       numElements;
} CIRCLIST_t;

// To initialize, set the begin and end pointers to the first and last entries
in the list
void initialize(CIRCLIST_t *var, LISTDATA_t *listStart, LISTDATA_t *listEnd);
void setDataToConstant(CIRCLIST_t *var, LISTDATA_t value);
void  addData(CIRCLIST_t *, LISTDATA_t); // add data to the list
UINT16 GetDataOldestFirst(LISTDATA_t *dest, const CIRCLIST_t *src, unsigned int
count); // copies data out of the buffer oldest to newest
UINT16 GetDataNewestFirst(LISTDATA_t *dest, const CIRCLIST_t *src, unsigned int
count); // copies data out of the buffer newest to oldest

#endif
```

```
#ifndef __event
#define __event

#include <main.h>
#include <string.h>

#define EVNT_HDR_VERSION   0x02

typedef struct {
  UINT16  event_header_version;
  EVENT_STATE   state;
  UINT32  guid;
  UINT8   id;
  char    time[20];    // string containing the timestamp for the event
  INT16   deltav[3];   // deltav for the event - used to replace old events
  INT16   peakG[3];    // peak accelerations for the events
  UINT16  headerBytes;
  UINT16  preTriggerBytes;
  UINT16  eventBytes;
  UINT32  preTriggerDataOffset;    // offset to pretriggerdata
  UINT32  dataOffset;              // offset to the start of data
  UINT32  baseAddr;                // address in flash of the event
  long    numerator[3];
  long    denominator[3];
  long    intercept[3];
  UINT16  zeros[3];
  UINT16  roll;
  UINT16  pitch;
  UINT16  yaw;
} EVENT_HEADER_t;

typedef struct {
  EVENT_STATE   state;
  UINT32        guid;
  UINT8         id;
  UINT32        baseAddr;              // address in flash of the event
} EVENT_DATA_t;

EVENT_HEADER_t  *EVENT_fetch(UINT32 addr);
EVENT_HEADER_t  *EVENT_write(void);
EVENT_HEADER_t  *EVENT_setconst(const UINT16 nums[3], const UINT16 dens[3],
const UINT16 ints[3], UINT16 roll, UINT16 pitch, UINT16 yaw);
EVENT_HEADER_t  *EVENT_setevent(const char *datetime, const UINT32 id, const
UINT16 zeros[3]);
EVENT_HEADER_t  *EVENT_setresults(const INT16 dv[3], const INT16 pg[3]);
EVENT_HEADER_t  *EVENT_setstate(EVENT_STATE newstate);
EVENT_HEADER_t  *EVENT_setaddr(UINT32 addr);
EVENT_HEADER_t  *EVENT_process(UINT32 addr, const UINT16 DV[], BOOL
(*EVENT_abort)(void));

#endif
```

```
#include <utility.h>

// This is the rotation matrix for UVW and XYZ data
const INT8 RotationMatrix[24][3] = {1    , 2    , 3,
                                    2    , -1   , 3,
                                    -1   , -2   , 3,
                                    -2   , 1    , 3,
                                    1    , -2   , -3,
                                    2    , 1    , -3,
                                    -1   , 2    , -3,
                                    -2   , -1   , -3,
                                    1    , 3    , -2,
                                    3    , -1   , -2,
                                    -1   ,-3    ,-2,
                                    -3   ,1     ,-2,
                                    1    ,-3    ,2,
                                    3    ,1     ,2,
                                    -1   ,3     ,2,
                                    -3   ,-1    ,2,
                                    2    ,-3    ,-1,
                                    -3   ,-2    ,-1,
                                    -2   ,3     ,-1,
                                    3    ,2     ,-1,
                                    2    ,3     ,1,
                                    -3   ,2     ,1,
                                    -2   ,-3    ,1,
                                    3    ,-2    ,1};

elem_type torben(elem_type m[], int n)
{
    int         i, less, greater, equal;
    elem_type  min, max, guess, maxltguess, mingtguess;

    min = max = m[0] ;
    for (i=1 ; i<n ; i++) {
        if (m[i]<min) min=m[i];
        if (m[i]>max) max=m[i];
    }

    while (1) {
        guess = (min+max)/2;
        less = 0; greater = 0; equal = 0;
        maxltguess = min ;
        mingtguess = max ;
        for (i=0; i<n; i++) {
            if (m[i]<guess) {
                less++;
                if (m[i]>maxltguess) maxltguess = m[i] ;
            } else if (m[i]>guess) {
                greater++;
                if (m[i]<mingtguess) mingtguess = m[i] ;
            } else equal++;
        }
        if (less <= (n+1)/2 && greater <= (n+1)/2) break ;
        else if (less>greater) max = maxltguess ;
        else min = mingtguess;
    }
    if (less >= (n+1)/2) return maxltguess;
```

```
    else if (less+equal >= (n+1)/2) return guess;
    else return mingtguess;
}

void shell_sort(UINT16 A[], UINT16 size)
{
  int i, j, incrmnt, temp;

  incrmnt = 3;
  while (incrmnt > 0)
  {
    for (i=0; i < size; i++)
    {
      j = i;
      temp = A[i];
      while ((j >= incrmnt) && (A[j-incrmnt] > temp))
      {
        A[j] = A[j - incrmnt];
        j = j - incrmnt;
      }
      A[j] = temp;
    }
    if (incrmnt>>1 != 0)
      incrmnt = incrmnt>>1;
    else if (incrmnt == 1)
      incrmnt = 0;
    else
      incrmnt = 1;
  }
}


/*
 * Algorithm from N. Wirth's book, implementation by N. Devillard.
 * This code in public domain.
 */

#define ELEM_SWAP(a,b) { register elem_type t=(a);(a)=(b);(b)=t; }
/*---------------------------------------------------------------------------
   Function :   kth_smallest()
   In       :   array of elements, # of elements in the array, rank k
   Out      :   one element
   Job      :   find the kth smallest element in the array
   Notice   :   use the median() macro defined below to get the median.

                Reference:

                  Author: Wirth, Niklaus
                   Title: Algorithms + data structures = programs
               Publisher: Englewood Cliffs: Prentice-Hall, 1976
    Physical description: 366 p.
                  Series: Prentice-Hall Series in Automatic Computation

 ---------------------------------------------------------------------------*/

elem_type kth_smallest(elem_type a[], int n, int k)
{
  register i,j,l,m ;
```

```
  register elem_type x ;

  l=0 ; m=n-1 ;
  while (l<m)
  {
    x=a[k] ;
    i=l ;
    j=m ;
    do
    {
      while (a[i]<x) i++;
      while (x<a[j]) j--;
      if (i<=j)
      {
          ELEM_SWAP(a[i],a[j]);
          i++;
          j--;
      }
    } while (i<=j) ;
    if (j<k) l=i ;
    if (k<i) m=j ;
  }
  return a[k] ;
}

UINT32  PackDeltaV(int uu, int vv, int ww)
{
  UINT32  ulPacked;
// Uplink the data to the MCM buffering the RX side
// Bounds check then pack and send
  CLIPTO(uu, 600, -600);
  CLIPTO(vv, 600, -600);
  CLIPTO(ww, 600, -600);
  ulPacked =
  (UINT32)(((UINT32)(uu+600))*1464100L+((UINT32)(vv+600))*1210L+((UINT32)(ww+60
  0))); // need 0.1 to pad data
  return ulPacked;
}

UINT8 AnglesToOrientation(int roll, int pitch, int yaw)
{
// Converts angles into an orientation number - look up table
  if ((roll==0)&&(pitch==0)&&(yaw==0)) return 1;
  if ((roll==0)&&(pitch==0)&&(yaw==270)) return 2;
  if ((roll==0)&&(pitch==0)&&(yaw==180)) return 3;
  if ((roll==0)&&(pitch==0)&&(yaw==90)) return 4;
  if ((roll==0)&&(pitch==180)&&(yaw==180)) return 5;
  if ((roll==0)&&(pitch==180)&&(yaw==90)) return 6;
  if ((roll==0)&&(pitch==180)&&(yaw==0)) return 7;
  if ((roll==0)&&(pitch==180)&&(yaw==270)) return 8;
  if ((roll==90)&&(pitch==180)&&(yaw==180)) return 9;
  if ((roll==90)&&(pitch==90)&&(yaw==180)) return 10;
  if ((roll==90)&&(pitch==0)&&(yaw==180)) return 11;
  if ((roll==90)&&(pitch==270)&&(yaw==180)) return 12;
  if ((roll==90)&&(pitch==0)&&(yaw==0)) return 13;
  if ((roll==90)&&(pitch==90)&&(yaw==0)) return 14;
  if ((roll==90)&&(pitch==180)&&(yaw==0)) return 15;
  if ((roll==90)&&(pitch==270)&&(yaw==0)) return 16;
```

```
  if ((roll==90)&&(pitch==0)&&(yaw==270)) return 17;
  if ((roll==90)&&(pitch==270)&&(yaw==270)) return 18;
  if ((roll==90)&&(pitch==180)&&(yaw==270)) return 19;
  if ((roll==90)&&(pitch==90)&&(yaw==270)) return 20;
  if ((roll==90)&&(pitch==180)&&(yaw==90)) return 21;
  if ((roll==90)&&(pitch==270)&&(yaw==90)) return 22;
  if ((roll==90)&&(pitch==0)&&(yaw==90)) return 23;
  if ((roll==90)&&(pitch==90)&&(yaw==90)) return 24;
  return 0;
}

ERROR_STATE UVWtoXYZ(UINT8 orient, INT16 *in, INT16 *out)
{
//---------------------------------------------------------------------------
// Rotates the data into the vehicle space based on the Witness orientation
// Uses the const rotation matrix to implement the change
//---------------------------------------------------------------------------
  int   i;
  INT16 lcl[3];

  lcl[0] = (in == 0) ? 0 : in[0];
  lcl[1] = (in == 0) ? 0 : in[1];
  lcl[2] = (in == 0) ? 100 : in[2];

  for (i=0;i<3;i++)
  {
    switch (RotationMatrix[orient-1][i])
    {
      case -3:
        out[i] = -lcl[2];
        break;
      case -2:
        out[i] = -lcl[1];
        break;
      case -1:
        out[i] = -lcl[0];
        break;
      case 1:
        out[i] = lcl[0];
        break;
      case 2:
        out[i] = lcl[1];
        break;
      case 3:
        out[i] = lcl[2];
        break;
    }
  }
}

ERROR_STATE XYZtoUVW(UINT8 orient, INT16 *in, INT16 *out)
{
//---------------------------------------------------------------------------
// Rotates the data from the vehicle space to the Witness orientation
// Uses the const rotation matrix to implement the change
//---------------------------------------------------------------------------
  int i;
```

```
  for (i=0;i<3;i++)
  {
    switch (RotationMatrix[orient-1][i])
    {
      case -3:
        out[2] = -in[i];
        break;
      case -2:
        out[1] = -in[i];
        break;
      case -1:
        out[0] = -in[i];
        break;
      case 1:
        out[0] = in[i];
        break;
      case 2:
        out[1] = in[i];
        break;
      case 3:
        out[2] = in[i];
        break;
    }
  }
}

UINT16 invert(UINT16 data, UINT16 reference)
{
  UINT16 diff;

  diff = (data > reference) ? data-reference : reference-data;
  return (data > reference) ? reference-diff : reference + diff;
}

ERROR_STATE RotateXYZtoUVW(UINT8 orient, UINT16 xyz[3], UINT16 uvw[3])
{
  int i;

  for (i=0;i<3;i++)
  {
    switch (RotationMatrix[orient-1][i])
    {
      case -3:
        uvw[2] = xyz[i];
        break;
      case -2:
        uvw[1] = xyz[i];
        break;
      case -1:
        uvw[0] = xyz[i];
        break;
      case 1:
        uvw[0] = xyz[i];
        break;
      case 2:
        uvw[1] = xyz[i];
        break;
      case 3:
```

```
        uvw[2] = xyz[i];
        break;
    }
  }
  return OK;
}
```

```
//-----------------------------------------------------------------------------
// This is the trigger prioritization list
//-----------------------------------------------------------------------------
#define SHUTDOWN       0b0000000000000001  // This is the lowest priority
#define TIMER_1HZ      0b0000000000000010  // This is the lowest priority
#define UART_TRIGGER   0b0000000000000100  // Message received
#define FLASH_EVENT    0b0000000000100000  // Indicates an event in the local
flash
#define TIMER_4HZ      0b0000000001000000
#define ADC_TRIG       0b0000000100000000  // Triggers below here will not
interrupt SPI flash operations
#define ADC1800_TRIG   0b0001000000000000
#define EVENT_TRIGGER 0b1000000000000000  // This is the highest priority

extern UINT16 glbTriggers;

#define SET_TRIGGER(TRIG) (glbTriggers |= TRIG)
#define CLR_TRIGGER(TRIG) (glbTriggers &= ~TRIG)
```

```
#include <accel.h>


const UINT8 test_pulse[325] =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,4,6,8,10,12,
14,16,18,20,22,24,26,28,30,32,34,36,38,40,
42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80,81,82,83,84,85,86,8
7,88,89,90,91,92,93,94,95,96,97,98,99,100,100,100,100,100,100,100,100,100,
99,99,98,98,97,96,96,96,95,95,94,94,93,93,92,92,91,91,90,90,90,89,89,88,88,87,8
7,86,86,85,85,85,84,84,83,83,82,82,81,81,80,80,80,79,79,78,78,77,77,76,76,
75,75,74,74,74,73,73,72,72,71,71,70,70,69,69,69,68,68,67,67,66,66,65,65,64,64,6
4,63,63,62,62,61,61,60,60,59,59,58,58,58,57,57,56,56,55,55,54,54,53,53,53,
52,52,51,51,50,50,49,49,48,48,48,47,47,46,46,45,45,44,44,43,43,42,42,42,41,41,4
0,40,39,39,38,38,37,37,37,36,36,35,35,34,34,33,33,32,32,32,31,31,30,30,29,
29,28,28,27,27,26,26,26,25,25,24,24,23,23,22,22,21,21,21,20,20,19,19,18,18,17,1
7,16,16,16,15,15,14,14,13,13,12,12,11,11,10,10,10,9,9,8,8,7,7,6,6,5,5,5,4,
4,3,3,2,2,1,1,1,1,1,1,1,1,1};

UINT16  updateTimeBase; // This is the calibrated update for the cleanest
signal

ERROR_STATE MMA1201_init(void)
{
  SETZTEST;
  __delay_cycles(20000);
  CLEARZTEST;
  return GETZSTATUS ? OK : FAIL;
}

ERROR_STATE MMA3201_init(void)
{
  SETXYTEST;
  __delay_cycles(20000);
  CLEARXYTEST;
  return GETXYSTATUS ? OK : FAIL;
}

ERROR_STATE MMA1201_self_test(BOOL state)
{
  if (state) SETZTEST; else CLEARZTEST;
}

ERROR_STATE MMA3201_self_test(BOOL state)
{
  if (state) SETXYTEST; else CLEARXYTEST;
}

ERROR_STATE MMA1201_pwm(PWM_STATE state, UINT16 base, UINT16 on)
{
  switch (state)
  {
    case INIT:
      PORT_AS_PWM;
// Configure the PWM output
      TBCTL = TBCLR | TBSSEL_2 | CNTL_0 | SHR_0 | MC_0 | ID_2;
      TBCCTL0 = 0;
      TBCCR0 = base;
      TBCCTL2 = OUTMOD_7;
```

```
      TBCCR2 = on;
      TBCTL |= MC_1; // start in the up mode
      break;
    case UPDATE_BASE:
      TBCTL = TBCLR | TBSSEL_2 | CNTL_0 | SHR_0 | MC_0 | ID_2;
      TBCCR0 = base;
      TBCTL |= MC_1; // start in the up mode
      break;
    case UPDATE_PWM:
      TBCTL = TBCLR | TBSSEL_2 | CNTL_0 | SHR_0 | MC_0 | ID_2;
      TBCCR2 = on;
      TBCTL |= MC_1; // start in the up mode
      break;
    case CLEAR:
      TBCTL = TBCLR | MC_0;
    case CLOSE:
      PORT_NORMAL;
      TBCTL = 0;
      TBCCTL2 = 0;
      break;
  }
}

ERROR_STATE MMA1201_test(void)
{
//-----------------------------------------------------------------------------
// Need to set up a PWM signal and send it to the sensor to simulate crash
//-----------------------------------------------------------------------------
  UINT16    i;
  UINT16    pwm;

  MMA1201_pwm(INIT, 450, 0);
// On each TBIFG - generate new pulse duration
  for (i=0; i< 319; i++)
  {
    pwm = test_pulse[i] << 2;
    while ((TBCTL & TBIFG) == 0) _NOP();     //
    MMA1201_pwm(UPDATE_PWM, 450, pwm);
    TBCTL &= ~TBIFG;                         // Clear the interrupt flag
  }
  MMA1201_pwm(CLOSE, 0, 0);
}

ERROR_STATE MMA3201_test(void)
{}
```

```
#include <event.h>
#ifndef NDEBUG
  #include <cross_studio_io.h>
#endif

EVENT_HEADER_t eventHeader;

EVENT_HEADER_t *EVENT_fetch(UINT32 addr)
{
  READ(&eventHeader, addr, sizeof(EVENT_HEADER_t));
  return &eventHeader;
}

EVENT_HEADER_t *EVENT_write(void)
{
  while (FlashBusy() == true) {}
  PP(eventHeader.baseAddr, &eventHeader, sizeof(EVENT_HEADER_t), true);
  return &eventHeader;
}

EVENT_HEADER_t *EVENT_setconst(const UINT16 nums[3], const UINT16 dens[3],
const UINT16 ints[3], UINT16 roll, UINT16 pitch, UINT16 yaw)
{
  UINT16  i;

  memset(&eventHeader, 0xFF, sizeof(EVENT_HEADER_t));
  eventHeader.event_header_version = EVNT_HDR_VERSION;
  eventHeader.headerBytes = sizeof(EVENT_HEADER_t);
  eventHeader.preTriggerBytes = BYTES_PER_PRETRIGGER;
  eventHeader.eventBytes = BYTES_PER_EVENT;
  eventHeader.preTriggerDataOffset = HEADER_ALLOCATION;    // offset to
  pretriggerdata
  eventHeader.dataOffset = ROUNDUP(eventHeader.preTriggerDataOffset +
  PRETRIGGER_ALLOC, PAGE_BYTES); // offset to the start of data
  for (i=0;i<3;i++)
  {
    eventHeader.numerator[i] = nums[i];
    eventHeader.denominator[i] = dens[i];
    eventHeader.intercept[i] = ints[i];
  }
  eventHeader.roll = roll;
  eventHeader.pitch = pitch;
  eventHeader.yaw = yaw;
  return &eventHeader;
}

EVENT_HEADER_t *EVENT_setevent(const char *datetime, const UINT32 id, const
UINT16 zeros[3])
{
//---------------------------------------------------------------------------
// starts a new event
//---------------------------------------------------------------------------
  UINT16  i;

  strncpy(eventHeader.time, datetime, 20);
  eventHeader.state = NEW;
  eventHeader.guid = id;
  eventHeader.id = (UINT8)(id & 0x000000FF);
```

```
  for (i=0;i<3;i++) eventHeader.zeros[i] = zeros[i];
  return &eventHeader;
}

EVENT_HEADER_t *EVENT_setresults(const INT16 dv[3], const INT16 pg[3])
{
  UINT16  i;

  for (i=0;i<3;i++)
  {
    eventHeader.deltav[i] = dv[i];
    eventHeader.peakG[i] = pg[i];
  }
  return &eventHeader;
}

EVENT_HEADER_t *EVENT_setstate(EVENT_STATE newstate)
{
  eventHeader.state = newstate;
  return &eventHeader;
}

EVENT_HEADER_t *EVENT_setaddr(UINT32 addr)
{
  eventHeader.baseAddr = addr;
  return &eventHeader;
}

EVENT_HEADER_t *EVENT_process(UINT32 addr, const UINT16 DV[], BOOL
(*EVENT_abort)(void))
{
  EVENT_HEADER_t *event;
  UINT16  buffer[96];
  UINT16  i,j,k;
  long    sums[3] = {0, 0, 0};
  int     sample;
  UINT16  absSample;
  UINT16  *q;

  event = EVENT_fetch(addr);
  if (event->event_header_version != EVNT_HDR_VERSION) return NULLP;
// Compute the parameters of interest
  addr = event->baseAddr + event->dataOffset;
  i = 0;
  while (i < SAMPLES_IN_200MSEC)
  {
    addr = READ(buffer, addr, 192);
    q = (UINT16 *)buffer;
    while (q < &buffer[96])
    {
      for (k=0;k<3;k++)
      {
        sample = *q++ - event->zeros[k];
        absSample = ABS16(sample);
        if (i == 0) event->peakG[k] = sample;
        if (absSample > ABS16(event->peakG[k])) event->peakG[k] = sample;
        sums[k] += sample;
        if (EVENT_abort()) return NULL;
```

```
      }
      i++;
    }
  } // <40msec to read data out
  for (i=0;i<3;i++)
  {
    event->peakG[i] = (event->peakG[i] * event->numerator[i] * 10) /
    event->denominator[i];
    event->deltav[i] = (MPH_PER_GS * sums[i] * event->numerator[i] * 10) /
    (event->denominator[i] * SAMPLE_RATE);
  }
// See if it is an event you want to keep
  event->state = BELOW;
  if (ABS16(event->deltav[0]) >= DV[0]) event->state = PROCESSED;
  if (ABS16(event->deltav[1]) >= DV[1]) event->state = PROCESSED;
  if (ABS16(event->deltav[2]) >= DV[2]) event->state = PROCESSED;

  return event;
}
```

```
#ifndef __fat
#define __fat

#include <eddef.h>
#include <st.h>

#define FAT_VERSION       2

//---------------------------------------------------------------------------
// This module handles the "file system" for storing data in the external flash
//---------------------------------------------------------------------------

typedef struct {
  UINT32  addr;          // address in flash to find the data
  BOOL    free;          // whether the entry is available
  UINT8   index;         // index into the event array corresponding to this
  sector
} FAT_ENTRY_t;

typedef struct {
  UINT8   fat_version;
  UINT32  blockSize;         // allocation for the event
  UINT8   free_entries;      // Number of free entries left in the system
  UINT8   sector_to_erase;   // This is the sector that gets erased to make way
  for new events
  UINT32  next_guid;         // This is the next number to assign from
  UINT32  sectorAddr[NUM_SECTOR];
  FAT_ENTRY_t fat[NUM_SECTOR][EVENTS_PER_SECTOR];
  UINT16  checksum; // make sure that FAT not corrupted
} FAT_DEFINITION_t;

FAT_DEFINITION_t   *w2FAT_build(void);
FAT_ENTRY_t        *w2FAT_findfree(void); // Returns the address for the next
open fat entry
ERROR_STATE        w2FAT_freeall(void);
UINT16             w2FAT_freemem(void);

#ifndef NDEBUG
  void ScanFAT(void);
#endif

#endif
```

```
#ifndef __crc
#define __crc

#include <eddef.h>

#define CRC16_POLY         0x8005
#define CRC16_INIT_REM     0x0
#define CRC16_FINAL_XOR    0x0

// CRC-16: x16 + x15 + x2 + 1
UINT16  UpdateChecksum(UINT16 crc, const char buf);
UINT16  ComputeChecksum(const void *data, UINT16 nBytes);
UINT16  XORChecksum(const void *data, UINT16 nBytes);

#endif
```

```
#include <uart.h>

UART_t myUART;
char *UARTout;

//---------------------------------------------------------------------------
// Communications routines
//---------------------------------------------------------------------------

void InitUART0(UINT16 baud)
{
  UART0_RX_IRQOFF;
  UARTout = myUART.OutputBuffer;
  myUART.RXIndex = myUART.TXIndex = myUART.nTXBytes = 0;
  UCTL0 = CHAR | SWRST;    // 8-bit character
  UTCTL0 = 0;              // UCLK = uiclk (1.8432MHz)
  switch (baud)
  {
    case 115200:
      UBR00 = 0x10;            // Configured for 38400
      UBR10 = 0x00;
      break;
    case 38400:
      UBR00 = 0x30;            // Configured for 38400
      UBR10 = 0x00;
      break;
    case 19200:
      UBR00 = 0x60;            // Configured for 19200
      UBR10 = 0x00;
      break;
    case 4800:
      UBR00 = 0x80;            // Configured for 38400
      UBR10 = 0x01;
      break;
  }
  UMCTL0 = 0x00;
  UCTL0 = CHAR;            // Release UART0
  ME1 = URXE0 | UTXE0;     // Enable USART0 RX and TX
  UART0_RX_IRQON;
}

void  SendBuffer(const void *buffer, const UINT8 length, BOOL newMsg)
{
  UINT8 i, *p;

  if (newMsg != false) myUART.CRC = CRC16_INIT_REM;
  p = (UINT8 *)buffer;
  for (i=0;i<length;i++)
  {
    myUART.OutputBuffer[i] = *p;
    myUART.CRC = UpdateChecksum(myUART.CRC, *p++);
  }
  myUART.TXIndex = 0;
  myUART.nTXBytes = length;
  myUART.Ready = false;
  U0TXBUF = myUART.OutputBuffer[0];
}
```

```
void  SendEvent(UINT8 id, unsigned long ulPacked)
{
  char    buf[14] = {0x41, 0x42, 0x43, 0x44, 0x00, 0x00, 0xEE, 0xEE, 0xEE,
  0xEE, 0x45, 0x46, 0x47, 0x48};
  buf[5] = id;
  ulPacked = __swap_long_bytes(ulPacked);
  memcpy(&buf[6], &ulPacked, 4);
// Change the baud rate and reset the UART to clear any pending transmissions
  UART0_TX_IRQON;
  SendBuffer(buf, 14, true);
  while (myUART.Ready == false) {} // wait for buffer register to clear
  UART0_TX_IRQOFF;
  while UART0_SHIFTING {} // hold reset until last character gone
}

ERROR_STATE SendDebugMessage(const char *str)
{
  myUART.nTXBytes = 0;
  UART0_TX_IRQON;
  SendBuffer(str, strlen(str), true);
  while UART0_SHIFTING {}
  UART0_TX_IRQOFF;
}

ERROR_STATE SendMCMLine(char *str)
{
  SendMCMMessage(str, strlen(str), START_MESSAGE);
  SendMCMMessage("", 0, CHECKSUM_ONLY);
}

ERROR_STATE SendMCMMessage(const void *str, UINT8 length, UINT8 code)
{
  char checksum[6];

  switch (code)
  {
    case START_MESSAGE:
      UART0_TX_IRQON;
      SendBuffer(str, length, true);
      while(UART0_BUSY) {}
      break;
    case CONTINUE_MESSAGE:
      SendBuffer(str, length, false);
      while(UART0_BUSY) {}
      break;
    case CLOSE_MESSAGE:
      SendBuffer(str, length, false);
      while(UART0_BUSY) {}
    case CHECKSUM_ONLY:
      sprintf(checksum, "\x04%04x", myUART.CRC);
      SendBuffer(checksum, strlen(checksum), true);
      while(UART0_SHIFTING) {}
      UART0_TX_IRQOFF;
      break;
    case CHECKSUM_NOSEP:
      sprintf(checksum, "%04x", myUART.CRC);
      SendBuffer(checksum, strlen(checksum), true);
      while(UART0_BUSY) {}
```

```
      while(UART0_SHIFTING) {}
      UART0_TX_IRQOFF;
      break;
    default:
      break;
  }
}

void usart0_rx (void) __interrupt[UART0RX_VECTOR]
{
  char  character;
  BOOL  addCharacter = true;

  character = U0RXBUF;
  switch (character)
  {
    case ACK:
      myUART.ACKreceived = myUART.replyReceived = true;
      addCharacter = false;
      break;
    case NAK:
      myUART.NAKreceived = myUART.replyReceived = true;
      addCharacter = false;
      break;
    case '*':
      memset(myUART.U0RXBuffer, 0x00, UART_BUF_LEN);
      myUART.RXIndex = 0;
      myUART.msgEnd = UART_BUF_LEN;
      break;
    case EOT:
      myUART.msgEnd = myUART.RXIndex + 4;
      break;
    case CR: case SPACE: case ':': case '/':
      break;
    default:
      if (isalnum(character) != 0) addCharacter = true;
      break;
  }
  if ((addCharacter != false) && (myUART.RXIndex < UART_BUF_LEN))
  {
    myUART.U0RXBuffer[myUART.RXIndex++] = character;
    if (myUART.RXIndex == myUART.msgEnd)
    {
      SET_TRIGGER(UART_TRIGGER);
      LPM4_EXIT;
    }
  }
}

void usart0_tx (void) __interrupt[UART0TX_VECTOR]
{
  if (myUART.nTXBytes == 0)
  {
    myUART.Ready = true;
    myUART.sentBytes = 0;
  }
  else
  {
```

```
    myUART.TXIndex++;
    if (myUART.TXIndex < UART_BUF_LEN)
    {
      U0TXBUF = myUART.OutputBuffer[myUART.TXIndex];
      if (myUART.TXIndex == (myUART.nTXBytes-1)) myUART.nTXBytes = 0;
    }
  }
}
```

```
#include <crc.h>

UINT16  UpdateChecksum(UINT16 crc, const char buf)
{
  unsigned int i, j;
  unsigned short msg;

  msg = (buf << 8);
  for(j = 0 ; j < 8 ; j++)
  {
    if((msg ^ crc) >> 15) crc = (crc << 1) ^ CRC16_POLY;
    else crc <<= 1;
    msg <<= 1;
  }
  return crc;
}

UINT16 ComputeChecksum(const void *data, UINT16 msg_size)
{
//---------------------------------------------------------------------------
// Ccomputes the checksum for the configuration and calibration data to assure
integrity
// Use a brute force algorithm since data blocks are quite small - no need for
// efficiency of table lookup. From TI documentation.
//---------------------------------------------------------------------------
  UINT16  crc = CRC16_INIT_REM;
  unsigned char *pmsg;
  unsigned int i, j;
  unsigned short msg;

  pmsg = (unsigned char *)data;
  for(i = 0 ; i < msg_size ; i ++)
  {
    msg = (*pmsg++ << 8);
    for(j = 0 ; j < 8 ; j++)
    {
      if((msg ^ crc) >> 15) crc = (crc << 1) ^ CRC16_POLY;
      else crc <<= 1;
      msg <<= 1;
    }
  }
  return(crc ^ CRC16_FINAL_XOR);
}

UINT16 XORChecksum(const void *data, UINT16 nBytes)
{
//---------------------------------------------------------------------------
// Simple and not strong checksum
//---------------------------------------------------------------------------
  UINT16  crc=0, *i, *start, *end;

  start = (UINT16 *)data;
  end = (UINT16 *)((UINT8 *)data + nBytes);
  for (i=start;i<end;i++) crc ^= *i;
  return crc;
}
```

```
#define P1_MSP2_RST_JTAG     BIT2
#define P1_MSP1_RST_JTAG     BIT3
#define P1_EVENT_SUPPRESS    BIT4
#define P1_FLASH2_RDY        BIT5
#define P1_MSP1_PWR          BIT6
#define P1_MSP2_PWR          BIT7
// ** Port 2
#define P2_J3_PRESENT    BIT0
#define P2_J2_PRESENT    BIT1
#define P2_MSP1_RST_JPR BIT2
#define P2_MSP2_RST_JPR BIT3
#define P2_TIRQ          BIT4
#define P2_AIRQ          BIT5
#define P2_MSP1_RST_OUT BIT6
#define P2_MSP2_RST_OUT BIT7
// ** Port 3
#define P3_AMUX0         BIT0
#define P3_AMUX1         BIT1
#define P3_AMUX_EN       BIT2
#define P3_UART_CLK      BIT3
#define P3_WITNESS_TX    BIT4
#define P3_WITNESS_RX    BIT5
#define P3_LED1          BIT6
#define P3_LED2          BIT7
// ** Port 4
#define P4_TEST_XY       BIT1
#define P4_TEST_Z        BIT2
#define P4_STATUS_XY     BIT3
#define P4_STATUS_Z      BIT4
#define P4_CS_ADC16      BIT5
#define P4_CS_MSP1       BIT6
#define P4_CS_MSP2       BIT7
// ** Port 5
#define P5_SPI_MASTER    BIT0
#define P5_SPI_MOSI      BIT1
#define P5_SPI_MISO      BIT2
#define P5_SPI_CLK       BIT3
#define P5_CS_FLASH0     BIT4
#define P5_CS_FLASH1     BIT5
#define P5_ANALOG_ON_1   BIT6
#define P5_ANALOG_ON_2   BIT7
// ** Port 6
#define P6_Z_DATA        BIT0
#define P6_Y_DATA        BIT1
#define P6_X_DATA        BIT2
#define P6_REF           BIT3

#define TIMERA0_OFF    TACCTL0 &= ~CCIE                    // CCR0 interrupt
enabled, output mode = toggle (TIMERA = 90Hz and 5Hz)
#define TIMERA0_ON     TACCTL0 |= CCIE                     // CCR0 interrupt
enabled, output mode = toggle (TIMERB = 1800Hz)
#define WATCHDOG_ON    WDTCTL = WDT_ARST_1000              // watchdog at 1000
msec.
#define WATCHDOG_OFF   WDTCTL = WDT_ARST_1000 | WDTHOLD    // stop watchdog (flash
operations)
#define EVENT_OFF      P1IE &= ~P1_EVENT
#define EVENT_ON       {P1IFG &= ~P1_EVENT; P1IE |= P1_EVENT;}
#define P2TIMER_OFF    P2IE &= ~P2_TIRQ
```

```
#ifndef __main
#define __main

#define FIRMWARE_REV                2
#define STATUS_REVISION             3
#define PRETRIGGER_SAMPLES          64 // Make sure that this is a power of
2!!!
#define TIMERA_ONE_SECOND           99
#define TIMERA_FIFTH_SECOND         19
#define PORT2_FIVE_SECOND           20
#define PORT2_ONE_SECOND            4
#define PORT2_FOURTH_SECOND         1
#define CCR_90HZ                    363
#define CCR_100HZ                   327
#define CCR_225HZ                   136
#define CCR_450HZ                   68
#define CCR_900HZ                   34
#define CCR_1800HZ                  17
#define HEADER_ALLOCATION           256
#define CHANNEL_COUNT               3
#define BYTES_PER_SAMPLE            (sizeof(UINT16) * CHANNEL_COUNT)
#define PRETRIGGER_ALLOC            512
#define SAMPLES_PER_EVENT           ((EVENT_ALLOCATION - HEADER_ALLOCATION -
PRETRIGGER_ALLOC) / BYTES_PER_SAMPLE)
#define SAMPLES_100MSEC_LEFT        (SAMPLES_PER_EVENT - 180)
#define BYTES_PER_EVENT             (SAMPLES_PER_EVENT * BYTES_PER_SAMPLE) //
compute from the integer number of samples that can be stored
#define BYTES_PER_PRETRIGGER        (PRETRIGGER_SAMPLES * BYTES_PER_SAMPLE)
#define SAMPLES_IN_200MSEC          360
#define BAUD_RATE                   115200

#define SAMPLE_RATE                 1800L
#define PRETRIG_RATE                100L
#define RATIO                       (SAMPLE_RATE / PRETRIG_RATE)
#define MPH_PER_GS                  22L

#include <__cross_studio_io.h>
#include <stdio.h>
#include <eddef.h>
#include <ctype.h>
#include <RTC.h>
#include <SPI.h>
#include <ST.h>
#include <fat.h>
#include <flash.h>
#include <event.h>
#include <crc.h>
#include <UART.h>
#include <lists.h>
#include <accel.h>
#include <utility.h>
#include <math.h>
#include <triggers.h>

// Define all of the port bits
// ** Port 1
#define P1_EVENT            BIT0
#define P1_DMM_EVENT        BIT1
```

```
#define P2TIMER_ON     {P2IFG &= ~P2_TIRQ; P2IE |= P2_TIRQ;}

#define MSP1_RESET_STATE     ((((P2IN & P2_MSP1_RST_JPR)==0) || (((P1IN &
P1_MSP1_RST_JTAG)==0) && (P2IN & P2_J2_PRESENT))) ? 0 : 1)
#define MSP2_RESET_STATE     ((((P2IN & P2_MSP2_RST_JPR)==0) || (((P1IN &
P1_MSP2_RST_JTAG)==0) && (P2IN & P2_J3_PRESENT))) ? 0 : 1)
#define MSP1_POWER_ON        { P2OUT &= ~P2_MSP1_RST_OUT; P1OUT |= P1_MSP1_PWR;
__delay_cycles(1000); P2OUT |= P2_MSP1_RST_OUT; }
#define MSP1_POWER_OFF       { P2OUT &= ~P2_MSP1_RST_OUT; P1OUT &= ~P1_MSP1_PWR;
}
#define MSP1_RESET           { P2OUT &= ~P2_MSP1_RST_OUT; __delay_cycles(1000);
P2OUT |= P2_MSP1_RST_OUT; }
#define MSP2_POWER_ON        { P2OUT &= ~P2_MSP2_RST_OUT; P1OUT |= P1_MSP2_PWR;
__delay_cycles(1000); P2OUT |= P2_MSP2_RST_OUT; }
#define MSP2_POWER_OFF       { P2OUT &= ~P2_MSP2_RST_OUT; P1OUT &= ~P1_MSP2_PWR;
}
#define MSP2_RESET           { P2OUT &= ~P2_MSP2_RST_OUT; __delay_cycles(1000);
P2OUT |= P2_MSP2_RST_OUT; }
#define POWER_SENSORS(VAL)   BITSET(P5OUT, P5_ANALOG_ON_1, VAL);
#define POWER_ANALOG(VAL)    BITSET(P5OUT, P5_ANALOG_ON_2, VAL);

#define SENSOR_STATUS 0x01

typedef struct {
  UINT8    Firmware_Rev;
  UINT8    Status_Structure_Revision;
  UINT8    Orientation;
  UINT16   roll;
  UINT16   pitch;
  UINT16   yaw;
  UINT8    Calibration_Status;
  UINT8    Percent_Memory_Available;
  UINT8    Diagnostic;
  char     Witness_ID[9];
  UINT16   GTrigger;
  UINT16   UVWDeltaV[3];
  UINT16   XYZDeltaV[3];
  UINT16   Numerator[3];
  UINT16   Denomenator[3];
  UINT16   Intercepts[3];
  char     Calibration_Date[20];
  char     RTC_Time[20];
  UINT16   checksum;
} Configuration_t;

typedef struct {
  BOOL     active;
  UINT8    index;
  UINT16   buffer[3][PRETRIGGER_SAMPLES];
} Pretrigger_t;

void         InitializeMSPPorts(void);    // Routine to initialize the ports for
the MSP430F149
void         ProcessMessages(char *msg, FAT_DEFINITION_t *fat);
void         CheckConfiguration(void);
ERROR_STATE DoCalibrate(const Configuration_t *config, const char *datetime);
BOOL         AcquireEvent(const char *t, UINT32 guid);
ERROR_STATE ProcessLocalFlash(void);
```

```
UINT16      MedianFilter(UINT16 *buffer, UINT16 nEntries);
ERROR_STATE ProcessEvents(const UINT16 DV[]);
void        SetSerialNumber(const char *sn);
void        SetOrientation(const char *params);
void        SetDeltaV(UINT16 x, UINT16 y, UINT16 z);
void        SetGTrigger(const char *params);
BOOL        ActivateReceived(const char *p, char *date);
BOOL        Event_Check(void);
void        ClearSystem(const void *config, const void *bufStart, const void
*bufend);
ERROR_STATE ValidConfiguration(const Configuration_t *config);
ERROR_STATE SetToDefaultConfiguration(const Configuration_t *config); // Check
the configuration information
void        SafeLocalFlashErase(const void *start, const void *end, BOOL
(*FLASH_abort)(void));
void        reload_event_list(FAT_DEFINITION_t *myFAT);

ERROR_STATE       FormatListHeader(const char *cmd, UINT8 fat_version);
ERROR_STATE       FormatEventSummaryList(const char *cmd);
ERROR_STATE       FormatTraceData(const char *cmd, UINT8 id, BOOL
(*abort)(void));
ERROR_STATE       processNew(const UINT16 DV[], BOOL (*abort)(void));
ERROR_STATE       sendNotifications(void);
ERROR_STATE       erase_event(UINT8 id);


#endif
```