```
////////////////////////////////////////////////////////////////////
//////////////////////
//
// Accelerometer Demonstration for Freescale DEMOQE Rev. C
Development Board
// -----------------------------------------------------------------
----------------------
//
// CodeWarrior V6.0 for MCUs
//
////////////////////////////////////////////////////////////////////
//////////////////////

#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "accelerometer.h" /* include main program defines and
declarations */

////////////////////////////////////////////////////////////////////
//////////////////////
// ICS_FEI
// -----------------------------------------------------------------
----------------------
// intitializes ICS for FEI mode with DCOH
////////////////////////////////////////////////////////////////////
//////////////////////
void ICS_FEI(void) {

  if (NVICSTRM != 0xFF)
    ICSTRM = NVICSTRM;                          // load trim value
if NV location not blank
  else
    ICSTRM = 0xAD;                  // use a default value if
NVICSTRM is blank
  ICSC1 = ICSC1_FEI;
  ICSC2 = ICSC2_FEI;
  ICSSC = ICSSC_FEI;
  while (ICSC1_CLKS != ICSSC_CLKST) {}  // wait for clk state to
match clk select
} //end InitICG

////////////////////////////////////////////////////////////////////
//////////////////////
// InitKBI
// -----------------------------------------------------------------
----------------------
// initializes 4 switches on DEMO or EVB as KBI's
////////////////////////////////////////////////////////////////////
//////////////////////
void InitKBI(void) {
// Enable KBI1P[3:2] as interrupt
  KBI1PE = KBI_SW;
```

```
  KBI1SC = 0b00000110;
/*                ||||
                  |||+---- KBIMOD = KBI detection mode: 0=edge only
                  ||+----- KBIE   = KBI int enable: 1=enabled
                  |+------ KBACK  = KBI int acknowledge: 1=clr IRQF
                  +------- KBF    = KBI flag
*/
}

////////////////////////////////////////////////////////////////////
//////////////////////
// InitSCI
// ----------------------------------------------------------------
----------------------
// initializes SCI1 to specified baudrate
////////////////////////////////////////////////////////////////////
//////////////////////
void InitSCI(word baud) {

  SCI1BD = baud;  // set baud
} //end InitSCI

////////////////////////////////////////////////////////////////////
//////////////////////
// RecChar & SendChar
// ----------------------------------------------------------------
----------------------
// receives/sends an ascii char on SCI1 at preset baudrate
////////////////////////////////////////////////////////////////////
//////////////////////
char RecChar(void) {
  byte rec_char;

  if (SCI1S1_RDRF)  // 1st half of RDRF clear procedure
    rec_char = SCI1D;  // 2nd half of RDRF clear procedure
  SCI1C2_RE = 1;    // enable Rx
  while(!SCI1S1_RDRF){  };
  rec_char = SCI1D; // get recieved character
  SendChar((char) rec_char); // echo received character
  return (char) SCI1D;
} //end RecChar

void SendChar(char s_char) {

  SCI1C2 = 0x08;    // enable Tx
  while(!SCI1S1_TDRE){ }
  SCI1D = (byte) s_char;   // 2nd half of TDRE clear procedure
} //end SendChar

////////////////////////////////////////////////////////////////////
//////////////////////
// SendMsg
```

```
// ------------------------------------------------------------------
----------------------
// sends an ascii string out SCI1 at preset baudrate
////////////////////////////////////////////////////////////////////
//////////////////////
void SendMsg(char msg[]) {
  byte i=0;
  char nxt_char;

  SCI1C2 = 0x08;    // enable Tx
  nxt_char = msg[i++];
  while(nxt_char != 0x00) {
    while(!SCI1S1_TDRE){}
    SCI1D = (byte) nxt_char; // 2nd half of TDRE clear procedure
    nxt_char = msg[i++];
  } //end while((SCI1D
} //end SendMsg

////////////////////////////////////////////////////////////////////
//////////////////////
// hex2bcd
// ------------------------------------------------------------------
----------------------
// converts hexadecimal word into a binary-coded decimal word
////////////////////////////////////////////////////////////////////
//////////////////////
word hex2bcd(word hex){
  byte dec[4],i;
  word bcd;


  for (i=0;i<4;i++){
    dec[i] = (byte) (hex%10);
    hex = (word) (hex/10);
  }

  if (hex>0){
    bcd=0xffff;
  }else{
    bcd=(word)((word)(dec[3]<<12) + (word)(dec[2]<<8) +
(dec[1]<<4) + dec[0]);
  }
  return bcd;
} //end hex2bcd

////////////////////////////////////////////////////////////////////
//////////////////////
// asc2byte & asc2word
// ------------------------------------------------------------------
----------------------
// converts an ascii string of 2 or 4 numeric chars into a byte or
word
```

```
////////////////////////////////////////////////////////////////
//////////////////////
byte asc2byte(char n_asc) {
  byte n;

  n = (byte)(n_asc - 0x30);      //convert from ascii to int
  if(n > 0x09)           // if num is $a or larger...
    n -= 0x07;           // ...sub $7 to correct
  if(n > 0x0f)           // if lower case was used...
    n -= 0x20;           // ...sub $20 to correct
  if(n > 0x0f)           // if non-numeric character...
    n = 0x00;            // ...default to '0'
  return n;
} //end asc2num

word asc2word(byte n_asc[2]) {
  word n,n2;

// assumes n_asc[0] is MSB, n_asc[1] is LSB
  n = (word)(n_asc[0] - 0x30);   //convert from ascii to int
  if(n > 0x09)           // if num is $a or larger...
    n -= 0x07;           // ...sub $7 to correct
  if(n > 0x0f)           // if lower case was used...
    n -= 0x20;           // ...sub $20 to correct
  if(n > 0x0f)           // if non-numeric character...
    n = 0x00;            // ...default to '0'
  n = (word)(n<<8);              // shift into high byte
  n2 = (word)(n_asc[1] - 0x30);  //convert from ascii to int
  if(n2 > 0x09)          // if num is $a or larger...
    n2 -= 0x07;          // ...sub $7 to correct
  if(n2 > 0x0f)          // if lower case was used...
    n2 -= 0x20;          // ...sub $20 to correct
  if(n2 > 0x0f)          // if non-numeric character...
    n2 = 0x00;           // ...default to '0'
  n += n2;               //
  return n;
} //end asc2word

////////////////////////////////////////////////////////////////
//////////////////////
// byte2asc & word2asc
// ---------------------------------------------------------------
----------------------
// converts a byte or word into an ascii string of 2 or 4 chars
////////////////////////////////////////////////////////////////
//////////////////////
char * byte2asc(byte num, byte base) {
  byte n;

  if (base){
    n=(byte)(hex2bcd(num));
  }else{
```

```
    n=num;
  } //end if (base)
  n_str[0] = (byte)((n>>0x04)+0x30);  // convert MSN to ascii
  if(n_str[0]>0x39)           // if MSN is $a or larger...
    n_str[0]+=0x07;           // ...add $7 to correct
  n_str[1] = (byte)((n&0x0f)+0x30);   // convert LSN to ascii
  if(n_str[1]>0x39)           // if LSN is $a or larger...
    n_str[1]+=0x07;           // ...add $7 to correct
  n_str[2] = 0x00;            // add line feed
  return  (char *) n_str;
} //end byte2asc

char * word2asc(word num, byte base) {
  word n;

  if (base){
    n=hex2bcd(num);
  }else{
    n=num;
  } //end if (base)

  n_str[0] = (byte)((n>>12)+0x30);    // convert MSN to ascii
  if(n_str[0]>0x39)           // if MSN is $a or larger...
    n_str[0]+=0x07;           // ...add $7 to correct
  n_str[1] = (byte)(((n>>8)&0x0f)+0x30);   // convert 2nd MSN to
ascii
  if(n_str[1]>0x39)           // if LSN is $a or larger...
    n_str[1]+=0x07;           // ...add $7 to correct
  n_str[2] = (byte)(((n>>4)&0x0f)+0x30);   // convert 2nd MSN to
ascii
  if(n_str[2]>0x39)           // if LSN is $a or larger...
    n_str[2]+=0x07;           // ...add $7 to correct
  n_str[3] = (byte)((n&0x0f)+0x30);   // convert 2nd MSN to ascii
  if(n_str[3]>0x39)           // if LSN is $a or larger...
    n_str[3]+=0x07;           // ...add $7 to correct
  n_str[4] = 0x00;    // add line feed
  return  (char *) n_str;

} //end word2asc

////////////////////////////////////////////////////////////////
//////////////////////
// StartTPM & StopTPM
// ----------------------------------------------------------------
----------------------
// Starts and stops TPM1 at busclk rate
////////////////////////////////////////////////////////////////
//////////////////////
void StartTPM(byte PS){
  TPM1SC = (byte)(0x08 | (0x07&PS));
  StartCount = TPM1CNT;
}// end StartTPM
```

```
word StopTPM(void){
  StopCount = (word)(TPM1CNT - StartCount);
  TPM1SC = 0;
  return StopCount;
}// end StopTPM

////////////////////////////////////////////////////////////////
//////////////////////
// PeriphInit
// ---------------------------------------------------------------
----------------------
// Initializes various registers and peripherals
////////////////////////////////////////////////////////////////
//////////////////////
void PeriphInit(void)
{
  // Disables COP and Enable STOP instruction and RESET and BKGD
pin
  SOPT1 = 0x23;

  // Selects FEI mode
  // Sets trimming for fBUS about 25 MHz
  ICS_FEI();

  // Enable all pullups
  PTAPE = 0xFF;
  PTBPE = 0xFF;
  PTCPE = 0xFF;
  PTDPE = 0xFF;
  PTEPE = 0xFF;
  PTFPE = 0xFF;
  PTGPE = 0xFF;
  PTHPE = 0xFF;
  PTJPE = 0xFF;

  /* Configures PTG[2:1] as accelerometer sensitivity
      PTG2:PTG1
        0    0  = 1.5g
        0    1  = 2.0g
        1    0  = 4.0g
        1    1  = 6.0g
  */
  PTGD = 0x00;
  PTGDD = 0x06;

  // Timer2 overflow about every 1ms
  TPM2MOD = 25000;
  // Stops timer2 and select 1 as prescaler divisor
  TPM2SC = 0x00;

  // Initializes SCI Peripheral
```

```
  InitSCI(fei_baud);

}

////////////////////////////////////////////////////////////////////////
//////////////////////
// filter_data
// ----------------------------------------------------------------------
----------------------
// Filters the collected x,y,z data using simple IIR filter
////////////////////////////////////////////////////////////////////////
//////////////////////
void filter_data(void)
{
  byte i;
  dword X, Y, Z;

  X = x.reading[samp];
  Y = y.reading[samp];
  Z = z.reading[samp];

  for (i=samp;i>0;i--){
    X = (X + ((x.reading[i] + x.result[i-1])>>1))>>1;
    Y = (Y + ((y.reading[i] + y.result[i-1])>>1))>>1;
    Z = (Z + ((z.reading[i] + z.result[i-1])>>1))>>1;
  }

  x.result[samp] = (word)X;
  y.result[samp] = (word)Y;
  z.result[samp] = (word)Z;
}// end filter_data

////////////////////////////////////////////////////////////////////////
//////////////////////
// avg_data
// ----------------------------------------------------------------------
----------------------
// - averages 10 collected x,y,z values
// - puts results in elements 0 of arrays
////////////////////////////////////////////////////////////////////////
//////////////////////
void avg_data(void)
{
  byte j;
  long x_avg=0, y_avg=0, z_avg=0;

  for (j=1;j<=samp;j++){
    x_avg += x.reading[j];
    y_avg += y.reading[j];
    z_avg += z.reading[j];
  }
  x.result[samp] = (word)(x_avg>>4);
```

```
  y.result[samp] = (word)(y_avg>>4);
  z.result[samp] = (word)(z_avg>>4);
}// end avg_data

////////////////////////////////////////////////////////////////////////
//////////////////////
// copy_data
// ---------------------------------------------------------------------
----------------------
// - copies reading into result
////////////////////////////////////////////////////////////////////////
//////////////////////
void copy_data(void) {

  x.result[samp] = x.reading[samp];
  y.result[samp] = y.reading[samp];
  z.result[samp] = z.reading[samp];
}// end copy_data

////////////////////////////////////////////////////////////////////////
//////////////////////
// ReadAcceleration
// ---------------------------------------------------------------------
----------------------
// Reads acceleration data on a given axis and saves it to the
axis structure
////////////////////////////////////////////////////////////////////////
//////////////////////
void ReadAcceleration(void){
  byte i;
  signed int temp;

  for(i=0;i<3;i++){
    temp = IIC_Rec_Data[i] & 0x3F;  //Get rid of Alert bit
    if(IIC_Rec_Data[i] & 0x20){
       temp |= 0xFFC0;                                  //Sign
extension
       temp += 32;
       IIC_Converted_Data[i] = temp;
    }else{
      IIC_Converted_Data[i] = temp + 32;
    }
  }
}//end ReadAcceleration

////////////////////////////////////////////////////////////////////////
//////////////////////
// ShowAcceleration
// ---------------------------------------------------------------------
----------------------
// -  Prints the accelaration data in the terminal;
```

```
//////////////////////////////////////////////////////////////////
//////////////////////
void ShowAcceleration (void)
{
  word SampleCNT;
  byte j,k;

  ReadAcceleration();            // Read acceleration data
  ADCSC1 = 0x01;                 // Select ADC1 (PTA1) channel
  x.reading[samp]  = (dword)( IIC_Converted_Data[0] <<8);
  ADCSC1 = 0x08;                 // Select ADC8 (PTA6) channel
  y.reading[samp]  = (dword)( IIC_Converted_Data[1] <<8);
  ADCSC1 = 0x09;                 // Select ADC9 (PTA7) channel
  z.reading[samp]  = (dword)( IIC_Converted_Data[2] <<8);

  StartTPM(0);   //0 = TPM prescaler = /2

  if(samp>0){
    switch (mode){
      case filter: filter_data();   break;
      case avg   : avg_data();      break;
      default    : copy_data();
    }
  } else {
    copy_data();
  }

  SampleCNT = StopTPM();
  if (SampleCNT<0x0100) {
    for(j=0xff;j>0;j--){
      for(k=0x10;k>0;k--){}
    }
  }

  // Display Acceleration
  SendMsg("\r\n");
  SendMsg(word2asc((word)x.result[samp],dis_base));
  SendMsg(",");
  SendMsg(word2asc((word)y.result[samp],dis_base));
  SendMsg(",");
  SendMsg(word2asc((word)z.result[samp],dis_base));
  SendMsg(",");
  SendMsg(word2asc(SampleCNT,dis_base));

  // Shift array of results if we hit max
  if (samp >= max-1) {
    for (j=0;j<max-1;j++){
      x.result[j]  = x.result[j+1];
      x.reading[j] = x.reading[j+1];
      y.result[j]  = y.result[j+1];
      y.reading[j] = y.reading[j+1];
      z.result[j]  = z.result[j+1];
```

```
      z.reading[j] = z.reading[j+1];
    }
    samp = max-1;
  } else {
    samp++;
  } //end if (i => max)

}// end ShowAcceleration

////////////////////////////////////////////////////////////////////////
//////////////////////
//  Function used as Master
//--------------------------------------------------------------------
----------------------
//  Master_Read_and_Store
//  Master_Write_MMA7660_register
//  Master_Read_MMA7660_register
////////////////////////////////////////////////////////////////////////
//////////////////////

void Master_Read_and_Store(void) {
  IIC_Rec_Data[rec_count++] = IIC2D;
}


void Master_Write_MMA7660_register(byte transbytes) {
  last_byte = 0;                    // Initialize variables to 0
     count = 0;
  bytes_to_trans = transbytes;

  if (transbytes == 0) return;

  IIC2C1_TX = 1;                    // Set TX bit for Address
cycle
  IIC2C1_MST = 1;                   // Set Master Bit to generate
a Start

  IIC2D = mma7660[count++];         // Send first byte (should be
7-bit address + R/W bit)
}

void Master_Read_MMA7660_register(byte transbytes, byte recbytes)
{

  rec_count = 0;                    // Initialize variables to 0
  last_byte = 0;
     count = 0;
     repeat_start_sent = 0;

  bytes_to_trans = transbytes;
  num_to_rec = recbytes;
```

```
  if (transbytes == 0) return;

  IIC2C1_TXAK = 0;
  IIC2C1_TX = 1;                    // Set TX bit for Address
cycle
  IIC2C1_MST = 1;                   // Set Master Bit to generate
a Start

  reading_mma7660_reg = 1;
  IIC2D = mma7660[count++];         // Send first byte (should be
7-bit address + R/W bit)
}

////////////////////////////////////////////////////////////////
//////////////////////
//  MMA7660_configuration
//--------------------------------------------------------------
----------------------
//  MMA7660 uses default configuration
//  120 samples/second; Disable interrupts
//  MMA7660 enter into active mode
////////////////////////////////////////////////////////////////
//////////////////////
void MMA7660_configuration(void){

  mma7660[0] = 0x98;
  mma7660[1] = 0x07;
  mma7660[2] = 0x01;
  Master_Write_MMA7660_register(3);

}
////////////////////////////////////////////////////////////////
//////////////////////
//  IIC_configuration
//--------------------------------------------------------------
----------------------
//  fIIC is 25MHz/4/48 = 130KHz
//  Enable IIC module and interrupts
////////////////////////////////////////////////////////////////
//////////////////////
void IIC_configuration (void) {

  IIC2F = 0x90;         /* Multiply factor of 4. SCL divider of 48
*/
  IIC2C1  = 0xC0;       /* Enable IIC module and interrupts */
}

////////////////////////////////////////////////////////////////
//////////////////////
// MAIN
```

```
// ------------------------------------------------------------------
----------------------
// Entry point
////////////////////////////////////////////////////////////////////
//////////////////////
void main(void){

  PeriphInit();
  InitKBI();
  IIC_configuration();

  EnableInterrupts;

  MMA7660_configuration();

// Selects fBUS as timer1 clock source and start timer
  TPM1SC = 0x08;
//  SendMsg("\fX, Y, Z\r\n");
  while (!SW1){}
  for(;;){
    while(SW4){
      if(!(IIC_Rec_Data[0]&IIC_Rec_Data[1]&IIC_Rec_Data[2]&0x40)){
        ShowAcceleration();         // Show acceleration data
      }else{                        // MMA7660 updating
      }

      if (PTHD_PTHD7 == 1) {        //Wait for IIC bus to be free

        while (PTHD_PTHD7 == 0);    // Wait while pin is low

           while (IIC2C1_MST == 1);    // Wait untill IIC is
stopped

           //Read Xout, Yout, Zout
           mma7660[0] = 0x98;
           mma7660[1] = 0x00;
           Master_Read_MMA7660_register(2,3);

      }else {

      }

    } //end while(SW4)
    while(SW3){_Stop;}
  } //end for(;;)

}

////////////////////////////////////////////////////////////////////
//////////////////////
// KBI_ISR
```

```
// ----------------------------------------------------------------------
// Reads PTA[3:2] and shifts to LSBs
// Debounces switch
// Acknowledges KBF
//////////////////////////////////////////////////////////////////////
interrupt VectorNumber_Vkeyboard void KBI_ISR(void){
  byte d,b;

  //capture which pin was pushed
  mode = (byte)(KBI_VAL);
  //debounce button
  for (d=0xff;d>0;d--){
    for (b=0x80;b>0;b--){}
  }
  //clear KBF
  KBI1SC_KBACK = 1;
}

//////////////////////////////////////////////////////////////////////
// IIC_ISR
// ----------------------------------------------------------------------
// IIC communication
// Master mode transmit and receive
//////////////////////////////////////////////////////////////////////

interrupt VectorNumber_Viicx void IIC_ISR(void) {

  IIC2S_IICIF = 1;              // Clear Interrupt Flag
  if (IIC2C1_TX) {              // Transmit or Receive?
////////////////// Transmit ////////////////////////////

    if (repeat_start_sent) {
      IIC2C1_TX = 0;                // Switch to RX mode
      if (num_to_rec == 1)
        IIC2C1_TXAK = 1;            // This sets up a NACK
            IIC2D;                          // Dummy read from Data
Register
    }
    else if ((last_byte) & (reading_mma7660_reg)) {
      IIC2C1_RSTA = 1;              //Repeat start
      IIC2D = (mma7660[0] | 0x01);  //Set Read bit
      repeat_start_sent = 1;
    }
          else if (last_byte) {           // Is the Last Byte?
               IIC2C1_MST = 0;              // Generate a Stop
          }
          else if (last_byte != 1) {
```

```
            if (IIC2S_RXAK) {              // Check for ACK
                  IIC2C1_MST = 0;              // No ACk Generate a
Stop
            }
            else if (!IIC2S_RXAK) {
                IIC2D = mma7660[count++];     // Transmit Data

                  if (count == bytes_to_trans)
                    last_byte = 1;
            }
          }
  } else {
//////////////////// Receive ////////////////////////////
    if ((num_to_rec - rec_count) == 2) {
            IIC2C1_TXAK = 1;          // This sets up a NACK
            Master_Read_and_Store();
    }
    else if ((num_to_rec - rec_count) == 1) {
            IIC2C1_MST = 0;           // Send STOP
            Master_Read_and_Store();
          }
          else {
          Master_Read_and_Store();
          }
  }
}
```