```c
/*!
 * \file    mpr084.c
 * \brief   Simple driver MPR084 using I2C
 * \version $Revision: 1.3 $
 * \author  Anthony Huereca
 */

 #include "mpr084.h"

/********************************** Variables **********************************/
unsigned char SlaveID;
unsigned char MasterTransmission;

volatile unsigned char keylog[8];
volatile unsigned char keypressed;

/*********************************************************************/
/*!
 * I2C Initialization
 * Set Baud Rate and turn on I2C
 */
void init_I2C(void)
{
  if(!PTGD_PTGD1) /* If connected to USB and thus 12MHz bus */
  {
    IIC1F  = 0x14;       /* set MULT and ICR */
  }
  else           /* 750kHz bus */
  {
    IIC1F  = 0x00;       /* set MULT and ICR */
  }
  IIC1C1 = 0x80;       /* enable IIC */
}

/*********************************************************************/
/*!
 * MPR084 Initialization
 */
void MPR084_init(void)
{
  /* Delay Loop to ensure MPR084 is powered up */
  int i;


  for(i=0;i<10000;i++) {
    asm("nop");
  }

  /* Turn on I2C Clock */
  SCGC1_IIC1=1;

  /* Configure I2C */
  init_I2C();

  /* Turn on ATTN Pin */
  PTCDD_PTCDD6=1;
  ATTN=1;

  /* Ensure MPR084 is awake */
  TOGGLE_ATTN

   /* Delay Loop to ensure MPR084 is powered up */
  for(i=0;i<10000;i++) {
    asm("nop");
  }

  /* Reset Touch Sensor */
  MPR084WriteRegister(CONFIGURATION_REGISTER,0x04);
  for(i=0;i<10000;i++) {
    asm("nop");
  }

  TOGGLE_ATTN

  for(i=0;i<10000;i++) {
    asm("nop");
```

```
  }

  /* Initialize key logger */
  for(i=0;i< LOG_LENGTH;i++)
  {
    keylog[i]=0;
  }

  /* Put in Stop 1 Mode for initial Config */
  MPR084WriteRegister(CONFIGURATION_REGISTER,0x14);

  /* Set maximum number of concurrent touches allowed to 1 */
  MPR084WriteRegister(MAX_TOUCH,0x01);

  /* Set Sensitivity Threshold to 0x3F */
  MPR084WriteRegister(SENSITIVITY_REGISTER1,0x3F);
  /* Set Sensitivity Threshold to 0x3F */
  MPR084WriteRegister(SENSITIVITY_REGISTER2,0x3F);
  /* Set Sensitivity Threshold to 0x3F */
  MPR084WriteRegister(SENSITIVITY_REGISTER3,0x3F);
  /* Set Sensitivity Threshold to 0x3F */
  MPR084WriteRegister(SENSITIVITY_REGISTER4,0x3F);
  /* Set Sensitivity Threshold to 0x3F */
  MPR084WriteRegister(SENSITIVITY_REGISTER5,0x3F);
  /* Set Sensitivity Threshold to 0x3F */
  MPR084WriteRegister(SENSITIVITY_REGISTER6,0x3F);
  /* Set Sensitivity Threshold to 0x3F */
  MPR084WriteRegister(SENSITIVITY_REGISTER7,0x3F);
  /* Set Sensitivity Threshold to 0x3F */
  MPR084WriteRegister(SENSITIVITY_REGISTER8,0x3F);

  /* Turn off Sounder */
  MPR084WriteRegister(SOUNDER_CONFIGURATION_REGISTER,0x00);

  /* Sound On, Auto Calibration On, Touch and Release Buffer On, Sensor On */
  MPR084WriteRegister(TOUCH_CONFIGURATION_REGISTER, 0x1D);

  /* Set 5ms Master Clock */
  MPR084WriteRegister(MASTER_TICK_COUNTER_REGISTER,0x00);

  /* TASP Multiplier to 2 */
  MPR084WriteRegister(TOUCH_ACQUISITION_SAMPLE_PERIOD_REGISTER,0x01);

  /* No Delay for Touches */
  MPR084WriteRegister(LOW_POWER_CONFIGURATION_REGISTER, 0x00);

  /* Turn on IRQ and put in Run 2 Mode */
  MPR084WriteRegister(CONFIGURATION_REGISTER,0x13);

  /* Turn off I2C Clock */
  SCGC1_IIC1=0;

  /* Turn on IRQ */
  IRQSC=0x52;
}


/*********************************************************************/
/*!
 * Read the Rotary Status Register to determine if have a touch
 * and if so, which button is touched
 * @return Status of Touch Pad
 */
byte ProximitySensorRead()
{
  byte result;

  /* Get current status of touch pad */
  result=u8MPR084ReadRegister(TOUCH_STATUS_REGISTER);


  /* Determine if a key is currently being touched */
  if(result&0xFF)
  {
    keypressed=1;

```

```c
    /* If key is touched, figure out which one it is... */
    switch(result&0xFF)
    {
        case 0x01:
          AddTouch(1);
          break;
        case 0x02:
          AddTouch(2);
          break;
        case 0x04:
          AddTouch(3);
          break;
        case 0x08:
          AddTouch(4);
          break;
        case 0x10:
          AddTouch(5);
          break;
        case 0x20:
          AddTouch(6);
          break;
        case 0x40:
          AddTouch(7);
          break;
        case 0x80:
          AddTouch(8);
          break;
        default:
          AddTouch(0);
          break;
    }
  }
  else
  {
    /* No key touched */
    keypressed=0;
  }
  return result;
}



/*********************************************************************/
/*!
 * IRQ Service Routine
 * Toggles the Attention pin to wake up MPR084 for I2C communications
 * Then reads Status register, clears FIFO buffer, and put back in low power mode
 * Only executes when button is either pushed or released
 */
interrupt VectorNumber_Virq void IRQ_ISR (void)
{
  byte fault;

  /* Turn on I2C Clock and Re-init I2C */
  SCGC1_IIC1=1;
  init_I2C();

  /* Put MPR084 into Run1 Mode for I2C Communication */
  TOGGLE_ATTN

  /* Turn off interrupts */
  MPR084WriteRegister(CONFIGURATION_REGISTER,0x15);

  /* Check for fault */
  fault=u8MPR084ReadRegister(FAULT_REGISTER);
  if(fault&0x00000007)
  {
    keypressed=0;
  }
  else
  {
    /* Read Sensor and Display on LED */
    ProximitySensorRead();
  }

```

```c
  /* Clear interrupt signal */
  while(IRQSC_IRQF)
  {
    MPR084WriteRegister(FAULT_REGISTER,0x00);
    u8MPR084ReadRegister(FIFO_REGISTER);

    IRQSC_IRQACK=1;
  }


  /* Put back in Low Power Mode */
  MPR084WriteRegister(CONFIGURATION_REGISTER,0x13);

  /* Turn off I2C Clock */
  SCGC1_IIC1=0;
}

/*********************************************************************/
/*!
 * Put the touch sensor into a low power mode. Drawback is that touches
 * take longer to register
 */
void MPR084SetLowPower()
{
  /* Turn on I2C Clock and Re-init I2C */
  SCGC1_IIC1=1;
  init_I2C();

  /* Ensure MPR084 is awake */
  TOGGLE_ATTN

  /* Put in Stop 1 Mode for initial Config */
  MPR084WriteRegister(CONFIGURATION_REGISTER,0x14);

  /* Sleep longer so require a long touch to wake up */
  MPR084WriteRegister(LOW_POWER_CONFIGURATION_REGISTER, 0x2F);

  /* Turn on IRQ and put in Run 2 Mode */
  MPR084WriteRegister(CONFIGURATION_REGISTER,0x13);

    SCGC1_IIC1=0;
}

/*********************************************************************/
/*!
 * Put the touch sensor into normal mode. Touches respond much faster,
 *  but draws about 400uA more current.
 */
void MPR084SetNormalPower()
{
  /* Turn on I2C Clock and Re-init I2C */
  SCGC1_IIC1=1;
  init_I2C();

  /* Ensure MPR084 is awake */
  TOGGLE_ATTN

  /* Put in Stop 1 Mode for initial Config */
  MPR084WriteRegister(CONFIGURATION_REGISTER,0x14);

  /* No Delay for Touches */
  MPR084WriteRegister(LOW_POWER_CONFIGURATION_REGISTER, 0x00);

  /* Turn on IRQ and put in Run 2 Mode */
  MPR084WriteRegister(CONFIGURATION_REGISTER,0x13);

    SCGC1_IIC1=0;

}


/*********************************************************************/
/*!
 * Return value of last key pressed
 * @return Number of the last button pressed
 */
```

```
305 byte LastKeyPressed()
306 {
307   return (byte)keylog[0] & 0x0000000F;
308 }
309
310 /*********************************************************************/
311 /*!
312  * Return 1 if key is currently being pressed. 0 if no key is being pressed
313  */
314 byte KeyPressed()
315 {
316   return keypressed & 0x0000000F;
317 }
318
319
320 /*********************************************************************/
321 /*!
322  * Return True if Slide Up on Left Side occured
323  */
324 byte SlideLeftUp()
325 {
326   return (keylog[0]==1) && (keylog[1]==2) && (keylog[2]==3) && (keylog[3]==4);
327 }
328
329 /*********************************************************************/
330 /*!
331  * Return True if Slide Down on Left Side occured
332  */
333 byte SlideLeftDown()
334 {
335   return (keylog[0]==4) && (keylog[1]==3) && (keylog[2]==2) && (keylog[3]==1);
336 }
337
338 /*********************************************************************/
339 /*!
340  * Return True if Slide Up on Right Side occured
341  */
342 byte SlideRightUp()
343 {
344   return (keylog[0]==5) && (keylog[1]==6) && (keylog[2]==7) && (keylog[3]==8);
345 }
346
347 /*********************************************************************/
348 /*!
349  * Return True if Slide Down on Right Side occured
350  */
351 byte SlideRightDown()
352 {
353   return (keylog[0]==8) && (keylog[1]==7) && (keylog[2]==6) && (keylog[3]==5);
354 }
355
356 /*********************************************************************/
357 /*!
358  * Return True if Slide Down on Right Side occured
359  */
360 byte ExitApp()
361 {
362   return (keylog[0]==8) && (keylog[1]==4) && (keylog[2]==5) && (keylog[3]==1);
363 }
364
365 /*********************************************************************/
366 /*!
367  *  Check for secret code
368  *  Return True if you read this
369  */
370 byte SecretCode(byte code)
371 {
372   switch (code)
373   {
374     case 2:
375       return ((keylog[6]==5) &&
376               (keylog[5]==6) &&
377               (keylog[4]==7) &&
378               (keylog[3]==8) &&
379               (keylog[2]==7) &&
380               (keylog[1]==6) &&
```

```
               (keylog[0]==5));
     case 1:
       return ((keylog[3]==1) &&
               (keylog[2]==5) &&
               (keylog[1]==2) &&
               (keylog[0]==6));
     case 0:
     default:
       return ((keylog[4]==3) &&
               (keylog[3]==1) &&
               (keylog[2]==3) &&
               (keylog[1]==3) &&
               (keylog[0]==7));
   }
}

/*********************************************************************/
/*!
 * Pause Routine
 */
void Pause(void){
    int n;
    for(n=1;n<50;n++) {
      asm("nop");
    }
}


/*********************************************************************/
/*!
 *  Add Key Press to Key History
 *  @param key is the key to add to queue
 */
void AddTouch(byte key)
{
  unsigned char i=LOG_LENGTH-1;

  /* Shift queue over by 1 */
  while(i>0)
  {
    keylog[i]=keylog[i-1];
    i--;
  }
  keylog[0]=key;

}


/*********************************************************************/
/*!
 * Start I2C Transmision
 * @param SlaveID is the 7 bit Slave Address
 * @param Mode sets Read or Write Mode
 */
void IIC_StartTransmission (unsigned char SlaveID,unsigned char Mode)
{
  if(Mode == MWSR)
  {
    /* set transmission mode */
    MasterTransmission = MWSR;
  }
  else
  {
    /* set transmission mode */
    MasterTransmission = MRSW;
  }

  /* shift ID in right possition */
  SlaveID = (byte) MPR084_I2C_ADDRESS << 1;

  /* Set R/W bit at end of Slave Address */
  SlaveID |= (byte)MasterTransmission;

  /* send start signal */
  i2c_Start();

```

```c
  /* send ID with W/R bit */
  i2c_write_byte(SlaveID);
}


/*********************************************************************/
/*!
 * Read a register from the MPR084
 * @param u8RegisterAddress is Register Address
 * @return Data stored in Register
 */
byte u8MPR084ReadRegister(byte u8RegisterAddress)
{
  byte result, result2;

  /* Set Register Pointer on MPR084 */
  IIC_StartTransmission(SlaveID,MWSR);
  i2c_Wait();

  IIC1D = u8RegisterAddress;
  i2c_Wait();

  i2c_Stop();

  Pause();

  /* Request data from Register */
  IIC_StartTransmission(SlaveID,MRSW);
  i2c_Wait();

  i2c_EnterRxMode();
  result2 = IIC1D;
  i2c_Wait();

  result = IIC1D;
  i2c_DisableAck();
  i2c_Wait();

  i2c_Stop();
  result2 = IIC1D;

  Pause();

  return result;
}

/*********************************************************************/
/*!
 * Write a byte of Data to specified register on MPR084
 * @param u8RegisterAddress is Register Address
 * @param u8Data is Data to write
 */
void MPR084WriteRegister(byte u8RegisterAddress, byte u8Data)
{
  /* send data to slave */
  IIC_StartTransmission(SlaveID,MWSR);
  i2c_Wait();

  IIC1D = u8RegisterAddress;
  i2c_Wait();

  IIC1D = u8Data;
  i2c_Wait();

  i2c_Stop();

  Pause();
}
```