

```
1  /*!
2  * \file      mpr084.c
3  * \brief     Simple driver MPR084 using I2C
4  * \version   $Revision: 1.3 $
5  * \author    Anthony Huereca
6  */
7
8  #include "mpr084.h"
9
10 /***** Variables *****/
11 unsigned char SlaveID;
12 unsigned char MasterTransmission;
13
14 volatile unsigned char keylog[8];
15 volatile unsigned char keypressed;
16
17 /*****
18  *!
19  * I2C Initialization
20  * Set Baud Rate and turn on I2C
21  */
22 void init_I2C(void)
23 {
24     if(!PTGD_PTGD1) /* If connected to USB and thus 12MHz bus */
25     {
26         IIC1F = 0x14; /* set MULT and ICR */
27     }
28     else /* 750kHz bus */
29     {
30         IIC1F = 0x00; /* set MULT and ICR */
31     }
32     IIC1C1 = 0x80; /* enable IIC */
33 }
34
35 /*****
36  *!
37  * MPR084 Initialization
38  */
39 void MPR084_init(void)
40 {
41     /* Delay Loop to ensure MPR084 is powered up */
42     int i;
43
44     for(i=0;i<10000;i++) {
45         asm("nop");
46     }
47
48     /* Turn on I2C Clock */
49     SCGC1_IIC1=1;
50
51     /* Configure I2C */
52     init_I2C();
53
54     /* Turn on ATTN Pin */
55     PTCDD_PTCDD6=1;
56     ATTN=1;
57
58     /* Ensure MPR084 is awake */
59     TOGGLE_ATTEN
60
61     /* Delay Loop to ensure MPR084 is powered up */
62     for(i=0;i<10000;i++) {
63         asm("nop");
64     }
65
66     /* Reset Touch Sensor */
67     MPR084WriteRegister(CONFIGURATION_REGISTER,0x04);
68     for(i=0;i<10000;i++) {
69         asm("nop");
70     }
71
72     TOGGLE_ATTEN
73
74     for(i=0;i<10000;i++) {
75         asm("nop");
76     }
```

```

77     }
78
79     /* Initialize key logger */
80     for(i=0;i< LOG_LENGTH;i++)
81     {
82         keylog[i]=0;
83     }
84
85     /* Put in Stop 1 Mode for initial Config */
86     MPR084WriteRegister(CONFIGURATION_REGISTER,0x14);
87
88     /* Set maximum number of concurrent touches allowed to 1 */
89     MPR084WriteRegister(MAX_TOUCH,0x01);
90
91     /* Set Sensitivity Threshold to 0x3F */
92     MPR084WriteRegister(SENSITIVITY_REGISTER1,0x3F);
93     /* Set Sensitivity Threshold to 0x3F */
94     MPR084WriteRegister(SENSITIVITY_REGISTER2,0x3F);
95     /* Set Sensitivity Threshold to 0x3F */
96     MPR084WriteRegister(SENSITIVITY_REGISTER3,0x3F);
97     /* Set Sensitivity Threshold to 0x3F */
98     MPR084WriteRegister(SENSITIVITY_REGISTER4,0x3F);
99     /* Set Sensitivity Threshold to 0x3F */
100    MPR084WriteRegister(SENSITIVITY_REGISTER5,0x3F);
101    /* Set Sensitivity Threshold to 0x3F */
102    MPR084WriteRegister(SENSITIVITY_REGISTER6,0x3F);
103    /* Set Sensitivity Threshold to 0x3F */
104    MPR084WriteRegister(SENSITIVITY_REGISTER7,0x3F);
105    /* Set Sensitivity Threshold to 0x3F */
106    MPR084WriteRegister(SENSITIVITY_REGISTER8,0x3F);
107
108    /* Turn off Sounder */
109    MPR084WriteRegister(SOUNDER_CONFIGURATION_REGISTER,0x00);
110
111    /* Sound On, Auto Calibration On, Touch and Release Buffer On, Sensor On */
112    MPR084WriteRegister(TOUCH_CONFIGURATION_REGISTER, 0x1D);
113
114    /* Set 5ms Master Clock */
115    MPR084WriteRegister(MASTER_TICK_COUNTER_REGISTER,0x00);
116
117    /* TASP Multiplier to 2 */
118    MPR084WriteRegister(TOUCH_ACQUISITION_SAMPLE_PERIOD_REGISTER,0x01);
119
120    /* No Delay for Touches */
121    MPR084WriteRegister(LOW_POWER_CONFIGURATION_REGISTER, 0x00);
122
123    /* Turn on IRQ and put in Run 2 Mode */
124    MPR084WriteRegister(CONFIGURATION_REGISTER,0x13);
125
126    /* Turn off I2C Clock */
127    SCGC1_IIC1=0;
128
129    /* Turn on IRQ */
130    IRQSC=0x52;
131 }
132
133
134 /*****
135  *!
136  * Read the Rotary Status Register to determine if have a touch
137  * and if so, which button is touched
138  * @return Status of Touch Pad
139  */
140 byte ProximitySensorRead()
141 {
142     byte result;
143
144     /* Get current status of touch pad */
145     result=u8MPR084ReadRegister(TOUCH_STATUS_REGISTER);
146
147
148     /* Determine if a key is currently being touched */
149     if(result&0xFF)
150     {
151         keypressed=1;
152     }

```

```
153     /* If key is touched, figure out which one it is... */
154     switch(result&0xFF)
155     {
156         case 0x01:
157             AddTouch(1);
158             break;
159         case 0x02:
160             AddTouch(2);
161             break;
162         case 0x04:
163             AddTouch(3);
164             break;
165         case 0x08:
166             AddTouch(4);
167             break;
168         case 0x10:
169             AddTouch(5);
170             break;
171         case 0x20:
172             AddTouch(6);
173             break;
174         case 0x40:
175             AddTouch(7);
176             break;
177         case 0x80:
178             AddTouch(8);
179             break;
180         default:
181             AddTouch(0);
182             break;
183     }
184 }
185 else
186 {
187     /* No key touched */
188     keypressed=0;
189 }
190 return result;
191 }
192
193
194
195
196 /*****
197  *!
198  * IRQ Service Routine
199  * Toggles the Attention pin to wake up MPR084 for I2C communications
200  * Then reads Status register, clears FIFO buffer, and put back in low power mode
201  * Only executes when button is either pushed or released
202  */
203 interrupt VectorNumber_Virq void IRQ_ISR (void)
204 {
205     byte fault;
206
207     /* Turn on I2C Clock and Re-init I2C */
208     SCGC1_IIC1=1;
209     init_I2C();
210
211     /* Put MPR084 into Run1 Mode for I2C Communication */
212     TOGGLE_ATTN
213
214     /* Turn off interrupts */
215     MPR084WriteRegister(CONFIGURATION_REGISTER,0x15);
216
217     /* Check for fault */
218     fault=u8MPR084ReadRegister(FAULT_REGISTER);
219     if(fault&0x0000007)
220     {
221         keypressed=0;
222     }
223     else
224     {
225         /* Read Sensor and Display on LED */
226         ProximitySensorRead();
227     }
228 }
```

```
229  /* Clear interrupt signal */
230  while(IRQSC_IRQF)
231  {
232      MPR084WriteRegister(Fault_Register,0x00);
233      u8MPR084ReadRegister(FIFO_Register);
234
235      IRQSC_IRQACK=1;
236  }
237
238
239  /* Put back in Low Power Mode */
240  MPR084WriteRegister(Configuration_Register,0x13);
241
242  /* Turn off I2C Clock */
243  SCGC1_IIC1=0;
244  }
245
246  /*****
247  /*!
248  * Put the touch sensor into a low power mode. Drawback is that touches
249  * take longer to register
250  */
251  void MPR084SetLowPower()
252  {
253      /* Turn on I2C Clock and Re-init I2C */
254      SCGC1_IIC1=1;
255      init_I2C();
256
257      /* Ensure MPR084 is awake */
258      TOGGLE_ATTN
259
260      /* Put in Stop 1 Mode for initial Config */
261      MPR084WriteRegister(Configuration_Register,0x14);
262
263      /* Sleep longer so require a long touch to wake up */
264      MPR084WriteRegister(Low_Power_Configuration_Register, 0x2F);
265
266      /* Turn on IRQ and put in Run 2 Mode */
267      MPR084WriteRegister(Configuration_Register,0x13);
268
269      SCGC1_IIC1=0;
270  }
271
272  /*****
273  /*!
274  * Put the touch sensor into normal mode. Touches respond much faster,
275  * but draws about 400uA more current.
276  */
277  void MPR084SetNormalPower()
278  {
279      /* Turn on I2C Clock and Re-init I2C */
280      SCGC1_IIC1=1;
281      init_I2C();
282
283      /* Ensure MPR084 is awake */
284      TOGGLE_ATTN
285
286      /* Put in Stop 1 Mode for initial Config */
287      MPR084WriteRegister(Configuration_Register,0x14);
288
289      /* No Delay for Touches */
290      MPR084WriteRegister(Low_Power_Configuration_Register, 0x00);
291
292      /* Turn on IRQ and put in Run 2 Mode */
293      MPR084WriteRegister(Configuration_Register,0x13);
294
295      SCGC1_IIC1=0;
296  }
297  }
298
299
300  /*****
301  /*!
302  * Return value of last key pressed
303  * @return Number of the last button pressed
304  */
```

```

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  * Return True if Slide Up on Left Side occurred
322  */
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 occurred
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 occurred
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 occurred
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 occurred
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) &&

```

```

381         (keylog[0]==5));
382     case 1:
383         return ((keylog[3]==1) &&
384             (keylog[2]==5) &&
385             (keylog[1]==2) &&
386             (keylog[0]==6));
387     case 0:
388     default:
389         return ((keylog[4]==3) &&
390             (keylog[3]==1) &&
391             (keylog[2]==3) &&
392             (keylog[1]==3) &&
393             (keylog[0]==7));
394     }
395 }
396
397 /*****
398  *!
399  * Pause Routine
400  */
401 void Pause(void){
402     int n;
403     for(n=1;n<50;n++) {
404         asm("nop");
405     }
406 }
407
408
409 /*****
410  *!
411  * Add Key Press to Key History
412  * @param key is the key to add to queue
413  */
414 void AddTouch(byte key)
415 {
416     unsigned char i=LOG_LENGTH-1;
417
418     /* Shift queue over by 1 */
419     while(i>0)
420     {
421         keylog[i]=keylog[i-1];
422         i--;
423     }
424     keylog[0]=key;
425 }
426
427
428
429 /*****
430  *!
431  * Start I2C Transmission
432  * @param SlaveID is the 7 bit Slave Address
433  * @param Mode sets Read or Write Mode
434  */
435 void IIC_StartTransmission (unsigned char SlaveID,unsigned char Mode)
436 {
437     if(Mode == MWSR)
438     {
439         /* set transmission mode */
440         MasterTransmission = MWSR;
441     }
442     else
443     {
444         /* set transmission mode */
445         MasterTransmission = MRSW;
446     }
447
448     /* shift ID in right possition */
449     SlaveID = (byte) MPR084_I2C_ADDRESS << 1;
450
451     /* Set R/W bit at end of Slave Address */
452     SlaveID |= (byte)MasterTransmission;
453
454     /* send start signal */
455     i2c_Start();
456

```

```
457     /* send ID with W/R bit */
458     i2c_write_byte(SlaveID);
459 }
460
461
462 /*****
463  *!
464  * Read a register from the MPR084
465  * @param u8RegisterAddress is Register Address
466  * @return Data stored in Register
467  */
468 byte u8MPR084ReadRegister(byte u8RegisterAddress)
469 {
470     byte result, result2;
471
472     /* Set Register Pointer on MPR084 */
473     IIC_StartTransmission(SlaveID,MWSR);
474     i2c_Wait();
475
476     IIClD = u8RegisterAddress;
477     i2c_Wait();
478
479     i2c_Stop();
480
481     Pause();
482
483     /* Request data from Register */
484     IIC_StartTransmission(SlaveID,MRSW);
485     i2c_Wait();
486
487     i2c_EnterRxMode();
488     result2 = IIClD;
489     i2c_Wait();
490
491     result = IIClD;
492     i2c_DisableAck();
493     i2c_Wait();
494
495     i2c_Stop();
496     result2 = IIClD;
497
498     Pause();
499
500     return result;
501 }
502
503 /*****
504  *!
505  * Write a byte of Data to specified register on MPR084
506  * @param u8RegisterAddress is Register Address
507  * @param u8Data is Data to write
508  */
509 void MPR084WriteRegister(byte u8RegisterAddress, byte u8Data)
510 {
511     /* send data to slave */
512     IIC_StartTransmission(SlaveID,MWSR);
513     i2c_Wait();
514
515     IIClD = u8RegisterAddress;
516     i2c_Wait();
517
518     IIClD = u8Data;
519     i2c_Wait();
520
521     i2c_Stop();
522
523     Pause();
524 }
525
526
527
```