```
 1 /*
 2  * File:   powerspy.c
 3  * Author: Manuel Federanko
 4  *
 5  * Created on 27 November 2015, 18:58
 6  *
 7  * All information regarding correlation between pins, registers and such
 8  * has been taken from microchips data sheet for PIC16(L)F1826/27 (DS41391D)
 9  */
10 
11 
12 #include <xc.h>
13 #include <limits.h>
14 #include "types.h"
15 #include "message.h"
16 #include "powerspy.h"
17 
18 //sin in scale of 0 to 100
19 __EEPROM_DATA(0, 2, 3, 5, 6, 8, 9, 11);
20 __EEPROM_DATA(13, 14, 16, 17, 19, 20, 22, 23);
21 __EEPROM_DATA(25, 26, 28, 29, 31, 32, 34, 35);
22 __EEPROM_DATA(37, 38, 40, 41, 43, 44, 45, 47);
23 __EEPROM_DATA(48, 50, 51, 52, 54, 55, 56, 58);
24 __EEPROM_DATA(59, 60, 61, 63, 64, 65, 66, 67);
25 __EEPROM_DATA(68, 70, 71, 72, 73, 74, 75, 76);
26 __EEPROM_DATA(77, 78, 79, 80, 81, 82, 83, 84);
27 __EEPROM_DATA(84, 85, 86, 87, 88, 88, 89, 90);
28 __EEPROM_DATA(90, 91, 92, 92, 93, 94, 94, 95);
29 __EEPROM_DATA(95, 96, 96, 96, 97, 97, 98, 98);
30 __EEPROM_DATA(98, 99, 99, 99, 99, 99, 100, 100);
31 __EEPROM_DATA(100, 100, 100, 100, 100, 0, 0, 0);
32 
33 const uint8_t get_shift_byte[10] = {NR0, NR1, NR2, NR3, NR4, NR5, NR6, NR7, NR8,
34 
35 
36 //remember interrupt time for deltat
37 uint16_t volt_time = 0;
38 uint16_t curr_time = 0;
39 
40 //0th bit: volts set
41 //1st bit: current set
42 //2nd bit: volts first
43 //3rd bit: current first
44 //4th bit: button
45 volatile uint8_t flag = 0;
46 
47 //config for current measurement
48 int24_t i_u_offs = -12500;
49 uint8_t i_u_diode_offs = 7;
50 
51 uint16_t led_rest = 0;
52 
53 volatile uint8_t mode = DMODE_NONE;
54 
55 /*****************************************************************************
56  *init methods*************************************************************
```

```
 57  ******************************************************************************/
 58
 59 /*
 60  * prepare the processor, nothing may be turned on
 61  * according to Pinlayout version 1.4
 62  */
 63 void initPins()
 64 {
 65         PORTA = 0b00000000;
 66         ANSELA = 0b00000111;
 67         TRISA = 0b00000111;
 68
 69         PORTB = 0b00000000;
 70         ANSELB = 0b00110000;
 71         TRISB = 0b00111010;
 72
 73         nWPUEN = 0;
 74         WPUB = 0b00001000;
 75
 76         //see data sheet page 65 & 58 FOSC set in pragmas
 77         //set the frequency to 32MHz
 78         SCS0 = 0;
 79         SCS1 = 0;
 80
 81         IRCF0 = 0;
 82         IRCF1 = 1;
 83         IRCF2 = 1;
 84         IRCF3 = 1;
 85
 86         SPLLEN = 1;
 87
 88         //config int for RB5 (see data sheet page 81)
 89         IOCBN5 = 1;
 90         IOCIE = 1;
 91 }
 92
 93 /*
 94  * set up the adc for an 8 bit conversion
 95  * with Vdd and Vss as references
 96  */
 97 void initADC()
 98 {
 99         //see page 139 of datasheet
100         //left justify, we have a resolution of 10 bit, 8 bit in ADRESH
101         ADFM = 0;
102
103         //see data sheet page 146
104         //set the conversion clock speed to FOSC/64
105         ADCS0 = 0;
106         ADCS1 = 1;
107         ADCS2 = 1;
108
109         //see data sheet page 146 || 139
110         //set the references to Vdd and Vss
111         ADPREF0 = 0;
112         ADPREF1 = 0;
113         ADNREF = 0;
```

```
114
115         ADON = 1;
116 }
117
118 /*
119  * this timer might be used as a screen refresch rate generator
120  * in a later release
121  */
122 void initTMR2()
123 {
124         /*
125          * prescaler vals for timers 2/4/6
126          * overflow_freq(*) = fosc_val/UCHAR_MAX
127          *
128          * *calculated with _XTAL_FREQ = 4_000_000
129          *
130          * |TxCKPS1|TxCKPS0|PRESCALER|FOSC_VAL|OVERFLOW_FREQ|RATE@50|
131          * |-------|-------|---------|--------|-------------|-------|
132          * |      0|      0|      1:1|  FOSC/4|       15_625|  312.5|
133          * |      0|      1|      1:4| FOSC/16|     3_906.25| 78.125|
134          * |      1|      0|     1:16| FOSC/64|      976.562|19.5312|
135          * |      1|      1|     1:64|FOSC/256|      244.141|4.88282|
136          *
137          */
138
139         //see data sheet page 189 & 191
140         //set the prescaler to 1:4 to get a rate of ~80
141         T2CKPS0 = 1;
142         T2CKPS1 = 0;
143
144         //see data sheet page 189 & 191
145         //set the postscaler to 1:1
146         T2OUTPS0 = 0;
147         T2OUTPS1 = 0;
148         T2OUTPS2 = 0;
149         T2OUTPS3 = 0;
150
151         //see data sheet page 190
152         //defaults to 0xff on reset, i want to be clear of what i want
153         PR2 = 0xff;
154
155
156         TMR2IE = 1;
157         TMR2IF = 0;
158
159         //see data sheet page 191
160         TMR2ON = 1;
161 }
162
163 /*
164  * used to measure real time for deltat and the led
165  */
166 void initTMR1()
167 {
168         /*
169          * see data sheet page 177
170          * set the clock source
```

```
171          * |TMR1CS1|TMR1CS0|              CLK SRC|
172          * |------:|------:|--------------------:|
173          * |      0|      0|                FOSC/4|
174          * |      0|      1|                  FOSC|
175          * |      1|      0|External CLK on T1CKI|
176          * |      1|      1|          Cap. S. OSC|
177          */
178
179         TMR1CS0 = 0;
180         TMR1CS1 = 0;
181
182         //set prescaler, ranges from 1 - 8
183         //we have a frequency of 32MHz with prescale of 2 (FOSC/8 = 4MHz) --> dt
184         T1CKPS0 = 1;
185         T1CKPS1 = 0;
186
187         TMR1IE = 1;
188         TMR1IF = 0;
189
190         TMR1ON = 1;
191 }
192
193 /*
194  * used to compare to voltage signal
195  */
196 void initFVR()
197 {
198         /*
199          * |CDAFVR1|CDAFVR0|VOLTS|
200          * |------:|------:|----:|
201          * |      0|      0|1.024|
202          * |      0|      1|2.048|
203          * |      1|      0|4.096|
204          * |      1|      1|    -|
205          */
206
207         //set the fvr for adc to 1.024 volts
208         ADFVR0 = 1;
209         ADFVR1 = 0;
210         //set FVR Buffer2 for comparing
211         CDAFVR0 = 0;
212         CDAFVR1 = 0;
213         FVREN = 1; //enable ref
214         while (!FVRRDY); //wait until its ready
215 }
216
217 //see data sheet page 208 and 209
218
219 /*
220  * used to compare amps signal
221  */
222 void initPWMTMR4()
223 {
224
225         //see data sheet page 119
226         CCP1SEL = 1; //switch output from RB3 to RB0
227
```

```
228         //1
229         TRISBbits.TRISB0 = 1; //disable output driver for RB0
230 
231         //2
232         PR2 = 0xff; //set duty cycle
233         CCP1CON = 0b00110000; //lsbs of duty cycle for 10 bit thingy, data sheet
234 
235         //3
236         CCP1CON |= 0b00001100;
237 
238         //4
239         //set the duty (1023 = 100%, 0 = 0%)
240         //0x7f + 1 + 1 == 511 = 50%
241         //8 higher bits
242         CCPR1L = 0x7f; //Weil besser.... kommen auf 2.48V
243 
244         //2 lower bits
245         DC1B0 = 1;
246         DC1B1 = 1;
247 
248         //5
249         //see data sheet page 227, tmr 4 is used
250         C1TSEL0 = 1;
251         C1TSEL1 = 0;
252         TMR4IF = 0;
253 
254         //see data sheet page 189
255         T4CKPS0 = 0;
256         T4CKPS1 = 0;
257 
258         TMR4ON = 1;
259 
260         //6
261         //while (!TMR4IF); //wait until overflow occured
262         TMR4IF = 0;
263 
264         TRISBbits.TRISB0 = 0;
265 }
266 
267 /*
268  * see data sheet page 164
269  * input for both comparators
270  * |C1NCH1|C1NCH0|    SRC|ON PIN|
271  * |-----:|-----:|------:|-----:|
272  * |     0|     0|C12IN0-|   RA0|
273  * |     0|     1|C12IN1-|   RA1|
274  * |     1|     0|C12IN2-|   RA2|
275  * |     1|     1|C12IN3-|   RA3|
276  */
277 
278 /*
279  * Comparator for phase measurement of the Voltage
280  * uses FVR Buffer2
281  */
282 void initCOMP1()
283 {
284         //select input channel
```

```
285         //see data sheet page 164
286         //C12IN1-
287         C1NCH0 = 1;
288         C1NCH1 = 0;
289 
290 
291         /*
292          * |C1PCH1|C1PCH0|     SRC|ON PIN|
293          * |-----:|-----:|-------:|-----:|
294          * |     0|     0|   C1IN+|   RA3|
295          * |     0|     1|     DAC|     -|
296          * |     1|     0|FVR BUF2|     -|
297          * |     1|     1|  C12IN+|   RA2|
298          */
299         //select compare source
300         //see data sheet page 164
301         //FVR BUF2
302         C1PCH0 = 0;
303         C1PCH1 = 1;
304 
305         //turn comp 1 on
306         C1INTP = 1; //falling edge, since the edges are reversed
307         C1IE = 1;
308         C1IF = 0;
309         C1ON = 1;
310 }
311 
312 /*
313  * Comparator for phase measurement of the Amps
314  * uses Voltage reference generated by the pwm
315  */
316 void initCOMP2()
317 {
318         //select input channel
319         //see data sheet page 165
320         //C12IN0-
321         C2NCH0 = 0;
322         C2NCH1 = 0;
323 
324         /*
325          * |C2PCH1|C2PCH0|     SRC|ON PIN|
326          * |-----:|-----:|-------:|-----:|
327          * |     0|     0|  C12IN+|   RA2|
328          * |     0|     1|     DAC|     -|
329          * |     1|     0|FVR BUF2|     -|
330          * |     1|     1|     VSS|     -|
331          */
332         //select compare source
333         //see data sheet page 165
334         //C12IN+
335         C2PCH0 = 0;
336         C2PCH1 = 0;
337 
338         //turn comp 2 on
339         C2INTP = 1; //falling edge, since the edges are reversed, actually it doe
340         C2IE = 1;
341         C2IF = 0;
```

```
342         C2ON = 1;
343 }
344
345 /*
346  * Init the USART module for communication with the bt module
347  * the bauderate is 9600
348  */
349 void initBT()
350 {
351
352         //select output pin
353         RXDTSEL = 0;
354         TXCKSEL = 0;
355
356         //Configure TX
357         TXEN = 1;
358         SYNC = 0;
359         SPEN = 1;
360
361         //Configure RX
362         CREN = 1;
363         SYNC = 0;
364         SPEN = 1;
365
366         //Set Baudrate for BT Module to 9600
367         BRGH = 0;
368         BRG16 = 0;
369         SPBRG = 51; //Datasheet Page 299
370
371         while (RCIF) {
372                 RCREG;
373         }
374
375         RCIE = 1;
376 }
377
378 /*****************************************************************************/
379 /*core methods*****************************************************************/
380 /*****************************************************************************/
381
382 /*
383  * give the desired channel <n>
384  * where n ranges from 0 to 11
385  *
386  * src: the desired channel
387  */
388 void adc(const int8_t src)
389 {
390         /*
391          * src is an char ranging from 0 to 11
392          *
393          * |SOURCE|REGISTER_PIN|EXTERNAL PIN/18|
394          * |------|------------|---------------|
395          * |AN0   |RA0         |17             |
396          * |AN1   |RA1         |18             |
397          * |AN2   |RA2         | 1             |
398          * |AN3   |RA3         | 2             |
```

```
399          * |AN4   |RA4         | 3             |
400          * |AN5   |RB6         |12             |
401          * |AN6   |RB7         |13             |
402          * |AN7   |RB5         |11             |
403          * |AN8   |RB4         |10             |
404          * |AN9   |RB3         | 9             |
405          * |AN10  |RB2         | 8             |
406          * |AN11  |RB1         | 7             |
407          * select the channel
408          * since the bit combination is sorted we can do this:
409          */
410
411         CHS0 = (bit) (src >> 0) & 0x01;
412         CHS1 = (bit) (src >> 1) & 0x01;
413         CHS2 = (bit) (src >> 2) & 0x01;
414         CHS3 = (bit) (src >> 3) & 0x01;
415         CHS4 = (bit) (src >> 4) & 0x01;
416
417         __delay_us(5);
418
419         //convert
420         GO_nDONE = 1;
421         while (GO_nDONE);
422         //the result is in ADRESH and ADRESL
423 }
424
425 /*****************************************************************************/
426 /*convenient methods*********************************************************/
427 /*****************************************************************************/
428
429 /*
430  * unused as of now, will probably not be implemented
431  */
432 uint8_t readVoltage()
433 {
434         return 230;
435 }
436
437 /*
438  * returns the current in amps as float
439  * since the voltage is regulated, we can just
440  * measure at any time
441  */
442 int24_t readCurrent()
443 {
444         /*
445          * function for current: curr(volt)=5*volt-12.5
446          * note: this only applies, when the processor is powered with 5Volts
447          * to convert from unitless to 0-5V we need to multiply ADRESH with 5
448          * and divide it thru 256 since we only use the 8 highest order bits
449          */
450         ADFM = 1;
451         adc(7);
452
453         //return (ADRES * 5000 / 1024 + i_u_diode_offs) * 5.0 + i_u_offs;
454         //measured val * value range / scale + offset
455         //return ((ADRES + i_u_diode_offs) * 5000) / 1024 + i_u_offs;
```

```
456         return (1000 * (5 * (ADRES + i_u_diode_offs) + i_u_offs)) / 1024;
457 }
458
459 uint16_t readVdd()
460 {
461         ADFM = 1;
462         adc(0b00011111); //fvr buffer output
463
464         //return (1024.0 / ADRES) * 1.024;
465         // flt_vdd = (1024.0/ ADRES) * 1.024
466         // int1000 = (1024 * 1000) / ADRES
467         //return (1024000) / ADRES
468         return ADRES<<2;
469 }
470
471 /*
472  * shift a char of data out of the pic
473  * the latch is not handled
474  * datapin: DISPLAY_DATA
475  * clockpin: DISPLAY_CLK
476  */
477 void so(const uint8_t data)
478 {
479         uint8_t c;
480         for (c = 0; c < CHAR_BIT; c++) {
481                 DISPLAY_DATA = (data >> c) & 0x01;
482                 SHIFT_DELAY
483                 SHIFT_DELAY
484                 SHIFT_DELAY
485                 SHIFT_DELAY
486                 DISPLAY_CLK = 1;
487                 SHIFT_DELAY
488                 SHIFT_DELAY
489                 SHIFT_DELAY
490                 SHIFT_DELAY
491                 DISPLAY_CLK = 0;
492         }
493 }
494
495 /*
496  * well, clears the display
497  */
498 void clearDisplay(int8_t leng)
499 {
500         for (; leng >= 0; leng--)
501                 so(0xff);
502 }
503
504 /*
505  * sends one byte of colour information to the led
506  * - highest bit first
507  */
508 void sendColour(uint8_t c)
509 {
510         if (c & 0b10000000) LED_HIGHBIT
511         else LED_LOWBIT;
512
```

```
513         if (c & 0b01000000) LED_HIGHBIT
514         else LED_LOWBIT;
515 
516         if (c & 0b00100000) LED_HIGHBIT
517         else LED_LOWBIT;
518 
519         if (c & 0b00010000) LED_HIGHBIT
520         else LED_LOWBIT;
521 
522         if (c & 0b00001000) LED_HIGHBIT
523         else LED_LOWBIT;
524 
525         if (c & 0b00000100) LED_HIGHBIT
526         else LED_LOWBIT;
527 
528         if (c & 0b00000010) LED_HIGHBIT
529         else LED_LOWBIT;
530 
531         if (c & 0b00000001) LED_HIGHBIT
532         else LED_LOWBIT;
533 }
534 
535 /*************************************************************************
536  *main program***********************************************************
537  ************************************************************************/
538 
539 /*
540  * compute the time difference, is only invoked by the interrupt service routine
541  * the flag byte should be cleared afterwards
542  */
543 uint16_t deltaT(uint16_t tm_low, uint16_t tm_high)
544 {
545         if (tm_low < tm_high) //no overflow bc obv
546                 return tm_high - tm_low;
547         else //no overflow bc now tm_low >= tm_high
548                 return 0xffff - tm_low + tm_high;
549 }
550 
551 int8_t sin_(int8_t z)
552 {
553         return eeprom_read(z);
554 }
555 
556 //z is in the range of 0 ... 100
557 
558 int8_t sin(int16_t z)
559 {
560         //us /= 50; //convert from us to winkel grad (not deg)
561         int16_t buff;
562         while (z > FULL_ROTATION)
563                 z -= FULL_ROTATION;
564 
565         if (z > (HALF_ROTATION) + QUARTER_ROTATION) { //4th quad
566                 buff = FULL_ROTATION;
567                 buff -= z;
568                 return -sin_((int8_t) buff);
569         }
```

```
570         if (z > HALF_ROTATION) {//3rd quad
571                 buff = z;
572                 buff -= HALF_ROTATION;
573                 return -sin_((int8_t) buff);
574         }
575         if (z > QUARTER_ROTATION) {//2nd quad
576                 buff = HALF_ROTATION;
577                 buff -= z;
578                 return sin_((int8_t) buff);
579         }
580 
581         //1st quad
582         return sin_((int8_t) z);
583 }
584 
585 int8_t cos(int16_t us)
586 {
587         return sin(QUARTER_ROTATION + us);
588 }
589 
590 uint8_t ledReset()
591 {
592         //because we have a prescale of 1:2 we can use only half of the required
593         if (led_rest < getTime()) {
594                 if (getTime() - led_rest > 200) return 1;
595                 else return 0;
596         } else if (led_rest > getTime()) {
597                 if (0xffff - led_rest + getTime() > 200) return 1;
598                 else return 0;
599         }
600         return 0;
601 }
602 
603 /*
604  * reprograms the led to the desired colour
605  */
606 void setLED(uint8_t g, uint8_t r, uint8_t b)
607 {
608         sendColour(g);
609         sendColour(r);
610         sendColour(b);
611         led_rest = getTime();
612 }
613 
614 void setUnit(uint24_t u)
615 {
616         so(u >> 16 & 0xff);
617         so(u >> 8 & 0xff);
618         so(u & 0xff);
619 }
620 
621 void setVal(int16_t v)
622 {
623         int8_t i;
624         for (i = 0; i < SHIFT_REG_LEN - 3; i++) {
625                 so(get_shift_byte[v % 10]);
626                 v /= 10;
```

```
627         }
628 }
629
630 void __interrupt ISR()
631 {
632         //usart data received
633         if (RCIE && RCIF) {
634                 receive_buff[buffpos] = RCREG;
635                 buffpos++;
636                 buffpos %= RECEIVEBUFF_SIZE;
637                 RCIF = 0;
638         }
639
640         //in us
641         if (TMR1IE && TMR1IF) {
642                 TMR1IF = 0;
643         }
644
645         //volts
646         if (C1IE && C1IF) {
647                 if (!(flag & 0x01)) { //is volt not set
648                         volt_time = getTime();
649                         if (flag & 0x02) //is current set
650                                 flag |= 0x08;
651                         flag |= 0x01;
652                 }
653
654                 C1IF = 0;
655         }
656
657         //amps
658         if (C2IE && C2IF) {
659                 if (!(flag & 0x02)) { //is current not set
660                         curr_time = getTime();
661                         if (flag & 0x01) //is volt set
662                                 flag |= 0x04;
663                         flag |= 0x02;
664                 }
665
666                 C2IF = 0;
667         }
668
669         if (TMR2IE && TMR2IF) {
670                 TMR2IF = 0;
671         }
672 }
673
674 /*
675  * main method, guess what it does!
676  */
677 void main()
678 {
679         int bmode = DMODE_NONE;
680         int24_t angle;
681         int24_t current;
682         int24_t voltage;
683         int24_t apparent;
```

```
684         int24_t real;
685         int24_t reactive;
686         int8_t i;
687 
688         PEIE = 0;
689         GIE = 0;
690 
691         initPins();
692         initFVR();
693         initADC();
694         initTMR1();
695         initPWMTMR4();
696         initCOMP1();
697         initCOMP2();
698         initBT();
699         initMessaging();
700         PEIE = 1;
701 
702         clearDisplay(SHIFT_REG_LEN);
703 
704         while (1) {
705 
706                 GIE = 1;
707                 __delay_ms(50);
708                 GIE = 0;
709 
710 
711                 //calculate the values and send only of both phases have been rec
712                 if ((flag & 0x02) && (flag & 0x01)) { //volts and current
713                         if (flag & 0x04) //volts first
714                                 angle = (deltaT(volt_time, curr_time) >> 2); // (
715                         else if (flag & 0x08) //current first
716                                 angle = (deltaT(curr_time, volt_time) >> 2);
717 
718                         flag &= 0xf0;
719 
720                         //angle now in us
721 
722                         //d = -5/2 vdd
723                         i_u_offs = readVdd() * 5;
724                         i_u_offs >>= 1;
725                         i_u_offs = -i_u_offs;
726 
727                         sendUInt8(K_RAWVOLTAGE);
728                         sendInt24(ADRES);
729 
730                         current = readCurrent();
731                         voltage = readVoltage();
732                         apparent = voltage * current;
733                         real = (apparent * cos(angle)) / MAX_SIN_RES;
734                         reactive = (apparent * sin(angle)) / MAX_SIN_RES;
735 
736 
737                         sendUInt8(K_RAWCURRENT);
738                         sendInt24(ADRES);
739                         sendUInt8(K_OFFS);
740                         sendInt24(i_u_offs);
```

```
741                         sendUInt8(K_CURRENT);
742                         sendInt24(current);
743                         sendUInt8(K_APPARENTEPOWER);
744                         sendInt24(apparent);
745                         sendUInt8(K_REALPOWER);
746                         sendInt24(real);
747                         sendUInt8(K_REACTIVEPOWER);
748                         sendInt24(reactive);
749 
750                         DISPLAY_LAT = 0;
751                         switch (bmode) {
752                                 case DMODE_NONE:
753                                 default:
754                                         setUnit(UNIT_NONE);
755                                         setVal(0);
756                                         setLED(0x00, 0x00, 0x00); //out
757                                         break;
758                                 case DMODE_CURRENT:
759                                         setUnit(UNIT_A);
760                                         setVal(current / 1000);
761                                         setLED(0x00, LED_INTENSE, 0x00); //red
762                                         break;
763                                 case DMODE_REAL:
764                                         setUnit(UNIT_W);
765                                         setVal(real / 1000);
766                                         setLED(LED_INTENSE, 0x00, 0x00); //green
767                                         break;
768                                 case DMODE_APPARENT:
769                                         setUnit(UNIT_VA);
770                                         setVal(apparent / 1000);
771                                         setLED(0x00, 0x00, LED_INTENSE); //blue
772                                         break;
773                                 case DMODE_REACTIVE:
774                                         setUnit(UNIT_VA);
775                                         setVal(reactive / 1000);
776                                         setLED(LED_INTENSE, 0x00, LED_INTENSE); /
777                                         break;
778                                 case DMODE_VOLTAGE:
779                                         setUnit(UNIT_V);
780                                         setVal(voltage);
781                                         setLED(LED_INTENSE, LED_INTENSE, 0x00); /
782                                         break;
783                         }
784                         DISPLAY_LAT = 1;
785                 }
786 
787 
788                 if (!BUTTON && !(flag & 0x10)) { //button has not been set
789                         bmode++;
790                         bmode %= DMODE_MAX;
791 
792                         flag |= 0x10;
793                         __delay_ms(10);
794                 } else {
795                         flag &= ~0x10;
796                 }
797         }
```

```
798 }
```