```
/*********************************************************************
    GOTO MARK-X Motor Drive Universul Controler

    Compiler  : HI-TECH C
    CPU       : PIC16F648A
    Oscillator: external 4MHz
    1-2相励磁でのクロック計算を基本とする

    2009/9/6～
    Copyright(C) 2009,2011 by embeddedark
 *********************************************************************/
#include    <htc.h>

//__CONFIG(XT & WDTDIS & PWRTEN & BOREN & MCLREN & LVPDIS & UNPROTECT);
//__CONFIG(XT & WDTDIS & PWRTEN & BORDIS & MCLREN & LVPDIS & UNPROTECT);

//__delay_us(), __delay_ms()の制御
#ifndef _XTAL_FREQ
    #define _XTAL_FREQ 4000000 //PICのクロックをHzで設定
#endif

/*=======================================
    BUILD OPTION
  =======================================*/
#ifndef EXTCLK
#define EXTCLK  1               //外部水晶発振子で発振させる
#endif
#define RARVSFUNC               //STOPキーを停止ではなく半速逆転として使う
#define RVSSLOWCTL              //回転方向が逆転する時に停止動作を一定時間挟む
#define DBGMON                  //デバッグ情報をEEPROMに書き込む
#undef MODESW_COMP              //モードスイッチとしてコンプリメンタリタイプのスイッチを使う
#define OPTION_CHANGE           //キー操作でオプション機能設定操作を行う

#if (EXTCLK==1)
__CONFIG(XT & WDTDIS & PWRTEN & BORDIS & MCLRDIS & LVPDIS & PROTECT);
#else
__CONFIG(INTCLK & WDTDIS & PWRTEN & BORDIS & MCLRDIS & LVPDIS & PROTECT);
#endif

/*-------------------------------
    I/O Difine
  -------------------------------*/
#define HIGH_LEVEL  (1)
#define LOW_LEVEL   (0)
#define ON          (1)
#define OFF         (0)
#define OUTB_       RA0             // PIN17 MORTOR OUTPUT PORT B_
#define OUTA_       RA1             // PIN18 MORTOR OUTPUT PORT A_
#define OUTB        RA2             // PIN 1 MORTOR OUTPUT PORT B
#define OUTA        RA3             // PIN 2 MORTOR OUTPUT PORT A
#define IS_OUTB_(x) ((x)&0x01)
#define IS_OUTA_(x) ((x)&0x02)
#define IS_OUTB(x)  ((x)&0x04)
#define IS_OUTA(x)  ((x)&0x08)
//#define IN_MOTOR_TYPE RA5         // PIN 4 MCLR_ MORTOR TYPE(external pull up)  1: GOTO STD / 0: direct
#define IN_FAST     RA4             // FAST SPEED (external pull up)
#define IN_STOP     RA5             // STOP (external pull up)
#define IN_OSC2     RA6             // PIN15 OSC2
#define IN_OSC1     RA7             // PIN16 OSC1
#define IN_DIR      RB0             // DIRECTION

#define SET_TRISA   (0b11110000)    // RA0(LSB)～RA3:OUTPUT,RA4～RA7(MSB):INPUT
#define INIT_PORTA  (0b00000000)    // PORTA INITIAL OUTPUT DATA
#define SET_TRISB   (0b11111011)    // OUTPUT TX(RB2)

#define SET_CMCON   (0b00000111)    // PORTA comparator off
#define UART_RX     RB1             // PIN 7 UART RX
#define UART_TX     RB2             // PIN 8 UART TX

/*=============================================
    TIMER COUNTER
    ※パルスモータのパルス数は1-2相励磁基準
  =============================================*/
//内部クロック(1/4)でプリスケーラを1/1にした場合の計算
#define RATCNT(x,y) (unsigned short)((float)(_XTAL_FREQ/4) * ((float)(x) / (float)(y)) + 0.5f)

#define HSPEED      3           // 恒星時比増速時スピード倍数設定
#define STD_PULSE       (7459200)   //GOTO標準モータ((126 * 600 * 48 * 74) / 36)
#define DIRECT_PULSE    (3628800)   //24ステップ(1-2相励磁時48ステップ)1/600ギア内蔵パルスモータ(126 * 600 * 48)
#define GOTO50HzPulse   (8618400)   //GOTO交流式モータ西日本仕様((126*125*48*57) / 5)
#define STAR_RATE   (86164)     //平均恒星時(86164.091)
#define STARRATE    (5000)      //比例値設定基準値：恒星時の基準値

#define DIVINT(x,y) (((x)+((y)>>1))/(y))
#define RATECNV(x)  DIVINT(STARRATE*STAR_RATE,x)
#define MARKX_STD_CNT       (11551) //RATCNT(STAR_RATE,STD_PULSE)
#define MARKX_DIRECT_CNT    (23745) //RATCNT(STAR_RATE,DIRECT_PULSE)

#define KINGS_RATE  (86190)     // 天文ガイドインタラクティブの記事の値は86200
                                // でもインターネットサイトでは86190が採用されている例が多い
#define KINGSRATE   RATECNV(KINGS_RATE) //比例値表現でのキングスレート値

#define SUN_RATE    (86400)     //平均太陽日[秒]
#define SUNRATE     RATECNV(SUN_RATE)       //比例値表現での太陽時値

#define MOON_RATE   (89455)     //平均月時[秒]:この値はネット上で検索した値(恒星時の96.3499%)
                                //http://homepage3.nifty.com/kunihiko/astro/telescope/motordrive/motordrive.htm
#define MOONRATE    RATECNV(MOON_RATE)  //比例値表現での月時値

#define LANDSCAPE_STAR_MAG  (6)     //星景モード倍率(割)(6の場合は0.6倍の意味)
#define LANDSCAPE_STAR_RATE DIVINT(STAR_RATE*10,LANDSCAPE_STAR_MAG)
#define LANDSCAPESTARRATE   RATECNV(LANDSCAPE_STAR_RATE)

#define HMDATA(x)   ((x)>>24)
#define HLDATA(x)   ((x)>>16)
#define HDATA(x)    ((x)>>8)
#define LDATA(x)    ((x)&0xff)

/**
 * motor type
 *
 */
#define MOTOR_MARKX     (0)             // MOTORTYPE = GOTO STD MARK-X
#define MOTOR_DIRECT    (1)             // MOTORTYPE = DIRECT 1/600 MOTOR for GOTO MARK-X

/*==================================
    EEPROM DATA
  ==================================*/
#define INIT_REIJI_MODE 0               /* 0:1-2相励磁/1:2-2相励磁*/
#define INIT_MOTOR_TYPE MOTOR_DIRECT    /* 0:GOTO STANDARD/1:GOTO DIRECT */
/*-------------------------
    動作モード定義
    0 : REIJI_MODE(0:1-2/1:2-2)
    1 : MOTOR_TYPE(0～NUM_MOTOR_TYPE-1)
    2,3: COMET_TIMER →恒星時を5000とした時の比例数値
```

```
126:         ※符号なし16ビット値：5000->STAR RATE/0->STOP
127:         ※数値の意味はコメットトラッカー互換
128:         ※但し、コメットトラッカーはBCD値だがここではBIN値
129:     4-7 : ソフトタイマカウンタモニタ
130:   ----------------------*/
131: __EEPROM_DATA(INIT_REIJI_MODE,INIT_MOTOR_TYPE,LDATA(STARRATE),HDATA(STARRATE),0xff,0xff,0xff,0xff);
132:
133: /*---------------------------------------
134:     モータータイプタイマカウンタ定義テーブル
135:
136:     標準恒星時基準に対応したタイマカウンタ値
137:
138:      8, 9 : MOTOR-TYPE=0 : GOTO MARK-X STANDARD MOTOR
139:     10,11 : MOTOR-TYPE=1 : GOTO MARK-X DIRECT MOTOR
140:     12,13 : MOTOR-TYPE=2 : 予備
141:     14,15 : MOTOR=TYPE=3 : 予備
142:   ---------------------------------------*/
143: #define NUM_MOTOR_TYPE  2
144: __EEPROM_DATA(LDATA(MARKX_STD_CNT),HDATA(MARKX_STD_CNT),LDATA(MARKX_DIRECT_CNT),HDATA(MARKX_DIRECT_CNT),0,0,0,0);
145:
146: /*---------------------------------------
147:     モードスイッチカウンタ比率値
148:     恒星時を5000としたときの比例値
149:     この方式の場合、恒星時以外の最終的なタイマカウンタ値は誤差を伴うが、
150:     もともと、大気差や季節に応じて変る値なので演算誤差は許容する。
151:
152:     16,17 : キングスレート
153:     18,19 : 平均太陽時
154:     20,21 : 平均月時
155:     22,23 : 星景モード
156:   ---------------------------------------*/
157: __EEPROM_DATA(LDATA(KINGSRATE),HDATA(KINGSRATE),LDATA(SUNRATE),HDATA(SUNRATE),LDATA(MOONRATE),HDATA(MOONRATE),LDATA(LANDSCAPESTARRATE),HDATA(LANDSCAPE
    STARRATE));
158:
159: #ifdef DBGMON
160: /*-------------------------
161:     デバッグ情報モニタ
162:     24 : 入力スイッチモニタ
163:     25 : モードスイッチモニタ
164:     26,27 : タイマカウンタモニタ0
165:     28,29 : タイマカウンタモニタ1
166:     30,31 : タイマカウンタモニタ2
167:     32,33 : タイマカウンタモニタ3
168:     34,35 : タイマカウンタモニタ4
169:     36,37 : タイマカウンタモニタ5
170:     38,39 : タイマカウンタモニタ6
171:   -------------------------*/
172: __EEPROM_DATA(0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff);
173: __EEPROM_DATA(0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff);
174: #endif
175:
176: /* EEPROM DATA INDEX */
177: #define EEP_REIJI          0
178: #define EEP_MOTORTYPE      1
179: #define EEP_COMET_TIMER    2
180: #define EEP_MONITOR_SOFTTIMER   4
181: #define EEP_MOTOR_TBL      8
182: #define EEP_RATE_TBL       16
183: #define EEP_SW             24
184: #define EEP_MODESW         25
185: #define EEP_MONITOR_TIMER  26
186:
187: #ifdef DBGMON
188: #define EEP_TBL_LAST       39
189: #else
190: #define EEP_TBL_LAST       23
191: #endif
192: #define EEP_TBL_MAX     (EEP_TBL_LAST+1)
193:
194: #if 0
195: /*====== GOTO MARK-X STANDARD MORTOR ======*/
196: /* STAR=11551,KINGS=11556,SUN=11583,MOON=11989*/
197: __EEPROM_DATA(0x2d,0x1f,0x2d,0x24,0x2d,0x3f,0x2e,0xd5);
198: /* LANDSCAPE and STAR=23102,DEC=3xSTAR=3850*/
199: __EEPROM_DATA(0x5a,0x3e,0x0f,0x0a,0,0,0,0);
200:
201: /*====== GOTO MARK-X DIRECT MORTOR ======*/
202: /* STAR=23745,KINGS=23754,SUN=23809,MOON=24644*/
203: __EEPROM_DATA(0x5c,0xc1,0x5c,0xca,0x5d,0x01,0x60,0x44);
204: /* LANDSCAPE and STAR=47489,DEC=3xSTAR=7915*/
205: __EEPROM_DATA(0xb9,0x81,0x1e,0xeb,0,0,0,0);
206:
207: #endif
208: /*-------------------------------
209:     EEPROM DATA AREA
210:   -------------------------------*/
211: static unsigned char eepdata[EEP_TBL_MAX];
212:
213:
214: /*-------------------------
215:     softwer sw asignment
216:   -------------------------*/
217: #define STOP_BIT    3
218: #define FAST_BIT    4
219: #define DIR_BIT     5
220: #define STOP_MASK   (0x01<<STOP_BIT)    // readsw() return bit mask
221: #define FAST_MASK   (0x01<<FAST_BIT)    // readsw() return bit mask
222: #define DIR_MASK    (0x01<<DIR_BIT)     // readsw() return bit mask
223: #define IN_STOP_SW(x)   (((x)&STOP_MASK)>>STOP_BIT)
224: #define IN_FAST_SW(x)   (((x)&FAST_MASK)>>FAST_BIT)
225: #define IN_DIR_SW(x)    (((x)&DIR_MASK)>>DIR_BIT)
226:
227: #define IN_MODE_SW(x)   ((x)&0x07)  // ROTARY SW LOWER 3bit : readsw() return bit mask
228:
229: #define SET_T1CON   (0b00000000)    // T1CKPS=0,T1OSCEN=0,T1SYNC=0,TMR1CS=0,TMR1ON=0
230: #define INIT_PORTB  (0b00000001)    // KEY STROBE OFF and first phase pulse
231:
232: /* operation mode */
233: #define MODE_STAR       (0)
234: #define MODE_KINGS_RATE (1)
235: #define MODE_SUN        (2)
236: #define MODE_MOON       (3)
237: #define MODE_LANDSCAPE_STAR (4)
238: #define MODE_DEC        (5)
239: #define MODE_COMET      (6)
240: #define MODE_OPTION     (7)
241: #define CHANGE_REIJI        MODE_OPTION
242: #define CHANGE_MOTORTYPE    MODE_OPTION
243: #define CHANGE_COMET        MODE_OPTION
244:
245: #define SWDELAY         (10)            // SW入力時間間隔[ms]
246: #define CHATTER_CNT     (5)             // チャタリング除去監視回数
247: #define INIT_DELAY      (100)
248: #ifdef RVSSLOWCTL
249: #define RVS_DELAY_CNT   (30)            //回転方向が変わる時に停止させる時間
```

```
250: #endif
251: #ifdef RARVSFUNC
252: #define RARVSFUNC_DELAY_CNT (200)       //STOPキーで逆転させるまでの遅延時間
253: #endif
254: #define MAX_SHORT   (65535)
255:
256: static unsigned char motor_type = INIT_MOTOR_TYPE;
257: static unsigned char reiji_mode = INIT_REIJI_MODE;
258: static unsigned char directionsw = 0;   // 1 : REVERSE  / 0 : FORWORD
259: static unsigned char direction = 0;
260: static unsigned char fastsw = 0;        // 0 : NORMAL   / 1 : HIGH SPEED
261: static unsigned char fast = 0;
262: static unsigned char stopsw = 0;        // 0 : NORMAL   / 1 : STOP
263: static unsigned char stop = 0;
264: static unsigned char initphase = INIT_DELAY;    // 初期化中カウンタ：このカウンタが非０の間はパルスを出力しない
265: #ifdef RVSSLOWCTL
266: static unsigned char rvs_delay = 0;
267: #endif
268: #ifdef RARVSFUNC
269: static unsigned char stop_delay = 0;
270: #endif
271: static unsigned char remote_mode = 0;   // 0 : NORMAL / 0以外 : REMOTE
272: static long softtimer = 0;
273: static long softtimer_set = 0;
274: static int softtimer_cnt = 0;
275:
276: static unsigned short tcnt = 0;         // 設定対象タイマカウンタ値
277: static int phase_inc = 0;               // パルス出力時の位相変化加算値 0:stop/1:forword/-1:reverse
278: static unsigned char comet_tcnt_set = OFF;  // EEPROMへのタイマカウンタ値設定動作状態
279:
280: /*-------------------------------------------------------------
281:     OUTPUT PULSE PHASE DEFINE
282:     モータ規格上の正転(CW)での定義とする
283:     モータタイプがMOTOR_MARKXの時はCWが追尾上の正転となるが
284:     モータタイプがMOTOR_DIRECTの時はCCWが追尾上の正転となる
285:   -------------------------------------------------------------*/
286: /* 1-2相励磁用パルステーブル */
287: static unsigned char pulse_tbl_12[] = {
288:                             /*    RGRG--->KP4P15G Color*/
289:                             /*    YBBW--->PK43G Color*/
290:                             /*       __*/
291:                             /*0000ABAB*/
292:                             0b00001001,
293:                             0b00001000,
294:                             0b00001100,
295:                             0b00000100,
296:                             0b00000110,
297:                             0b00000010,
298:                             0b00000011,
299:                             0b00000001
300:                             };
301: /* 2-2相励磁用パルステーブル：1-2相励磁クロック基準なので8パターン定義*/
302: static unsigned char pulse_tbl_22[] = {
303:                             /*    RGRG--->KP4P15G Color*/
304:                             /*    YBBW--->PK43G Color*/
305:                             /*     rl */
306:                             /*       __*/
307:                             /*0000ABAB*/
308:                             0b00001001,
309:                             0b00001001,
310:                             0b00001100,
311:                             0b00001100,
312:                             0b00000110,
313:                             0b00000110,
314:                             0b00000011,
315:                             0b00000011
316:                             };
317: static unsigned char * pulse_ptbl[] = { pulse_tbl_12,pulse_tbl_22 };
318: static unsigned char * pulse_tbl;
319:
320: /*-------------------------------
321:     TIMER CNT DEFINE
322:   -------------------------------*/
323: static unsigned short * timer_tbl = (unsigned short *)&eepdata[EEP_MOTOR_TBL];
324: static unsigned short * rate_tbl = (unsigned short *)&eepdata[EEP_RATE_TBL];
325:
326: /**
327:  *  タイマ割り込みルーチン関数ポインタ設定
328:  */
329: void timer1_isr_direct(void);
330: void timer1_isr_softtimer(void);
331: static void (*timer1_isr)(void) = timer1_isr_direct;
332:
333: /*=========================================
334:     以下のE2promRead(),E2promWrite()は
335:     浜松マイコン工房さんの著作物です
336:     http://micom.hamazo.tv/e3140018.html
337:   =========================================*/
338: //---------------------------------------------------------------------
339: // Read data
340: //---------------------------------------------------------------------
341: unsigned char E2promRead( unsigned char adr )
342: {
343:     EEADR = adr;
344:     RD = 1; // Set read bit
345:     return( EEDATA ); // Return read data
346: }
347:
348:
349: //---------------------------------------------------------------------
350: // Write data
351: //---------------------------------------------------------------------
352: void E2promWrite( unsigned char adr, unsigned char data )
353: {
354:     EEADR = adr;
355:     EEDATA = data;
356:
357:     WREN = 1;   // Set write enable bit
358:     EECON2 = 0x55;
359:     EECON2 = 0xaa;
360:
361:     WR = 1;   // Set programming bit
362:     while( EEIF == 0 ) // Wait till write operation complete
363:     {
364:         NOP();
365:     }
366:     EEIF = 0; // Clear EEPROM write complete bit
367:     WREN = 0;   // Clear write enable bit
368: }
369:
370: /*=====================================
371:     EEPROMデータ読み込み
372:   =====================================*/
373: unsigned char eepread(unsigned char add)
374: {
```

```
375:     if( sizeof(eepdata)>add ) {
376:         return eepdata[add];
377:     }
378:     else {
379:         return E2promRead(add);
380:     }
381: }
382:
383: /*====================================
384:     EEPROMデータ書き込み処理
385:     ※リトライ処理をすると何かおかしいので廃止
386:   ====================================*/
387: void eepwrite(unsigned char add,unsigned char data)
388: {
389:     unsigned char dt = eepread(add);
390:
391:     if( dt != data ) {
392:         //現在のデータと違う時のみ実際の書き込みを行う
393:         E2promWrite(add,data);
394:     }
395:     if( sizeof(eepdata)>add ) {
396:         //キャッシュデータ更新
397:         eepdata[add] = data;
398:     }
399: }
400:
401: #define eepread_short(add)  ((unsigned short)eepread(add+1)<<8 | (unsigned short)eepread(add))
402: #define eepread_long(add)   ((unsigned long)eepread(add+3)<<24 | (unsigned long)eepread(add+2)<<16 | (unsigned long)eepread(add+1)<<8 | (unsigned
     long)eepread(add))
403: #define eepwrite_short(add,data) {¥
404:     eepwrite(add,LDATA(data));¥
405:     eepwrite((add)+1,HDATA(data));¥
406: }
407:
408: #define eepwrite_long(add,data) {¥
409:     eepwrite((add),LDATA(data));¥
410:     eepwrite((add)+1,HDATA(data));¥
411:     eepwrite((add)+2,HLDATA(data));¥
412:     eepwrite((add)+3,HMDATA(data));¥
413: }
414:
415: /**
416:  *  timer1 isr setup and softtimer setup
417:  */
418: unsigned short timer_isr_set(long cnt)
419: {
420:         di();
421:         if( MAX_SHORT<cnt ) {
422:             timer1_isr = timer1_isr_softtimer;
423:             softtimer_set = softtimer = cnt;
424:             for(softtimer_cnt = 2;MAX_SHORT<(cnt/softtimer_cnt);softtimer_cnt++);
425:             cnt /= softtimer_cnt;
426:         }
427:         else {
428:             timer1_isr = timer1_isr_direct;
429:             softtimer_set = softtimer = 0;
430:             softtimer_cnt = 0;
431:         }
432:         ei();
433:         return (unsigned short)cnt;
434: }
435:
436: /*----------------------------------
437:     get timer counter
438:   ----------------------------------*/
439: unsigned short timer_counter(unsigned char mode)
440: {
441:     unsigned long cnt;
442:
443:     if( MODE_OPTION==mode ) {
444:         //オプションで動作を割り当てていないのでカウンタ値は0にしておく
445:         cnt = 0;
446:     }
447:     else {
448:         cnt = timer_tbl[motor_type];    //モータ種別毎の恒星時カウンタ取得
449:
450:         if( MODE_STAR != mode ) {
451:             cnt *= STARRATE;
452:             if( MODE_COMET==mode ) {
453:                 //コメットモード時比率修正
454:                 unsigned short comet_set;
455:                 comet_set = eepread_short(EEP_COMET_TIMER);
456:                 if( 0==comet_set ) {
457:                     cnt = 0;
458:                 }
459:                 else {
460:                     cnt = DIVINT(cnt,comet_set);
461:                 }
462:             }
463:             else {
464:                 //コメットモード時以外
465:                 if( MODE_DEC==mode ) {
466:                     cnt = DIVINT(cnt,HSPEED*STARRATE);
467:                 }
468:                 else {
469:                     cnt = DIVINT(cnt,rate_tbl[mode-1]);
470:                 }
471:             }
472:         }
473:     }
474:     return timer_isr_set(cnt);
475: }
476:
477: /*========================================
478:         sw read
479:   ========================================*/
480: static unsigned char swport = 0;
481: unsigned char read_sw(int type)
482: {
483:     static unsigned char same = CHATTER_CNT;
484:     static unsigned char swport2 = 0;
485:     static unsigned char scanon = 0;    //チャタリング除去スキャン中フラグ
486:     unsigned char swport1;
487:
488:     /*================================
489:         DIRECT PORT CONTROL
490:       ================================*/
491:     __delay_ms(SWDELAY);    /* wait for loop */
492:
493:     //ポート全ビットを負論理入力
494:     swport1 = ~PORTB;       // 76 543  210
495:                             // xx MODE xxx
496:                             // MODE = DIP SW/ROTARY SW
497:     swport1 >>= 3;
498:
```

```
#ifdef MODESW_COMP          // コンプリメンタリタイプスイッチ
    swport1 = ~swport1;
#endif

    swport1 &= 0x07;

    //負論理ポート入力
    if( 0==IN_DIR )     swport1 |= DIR_MASK;
    if( 0==IN_STOP )    swport1 |= STOP_MASK;
    if( 0==IN_FAST )    swport1 |= FAST_MASK;

    if( -1 == type ) {
        /* チャタリングを気にせずに即座に値を返す */
        swport = swport1;
    }
    else if( scanon ) {
        if( swport1==swport2 ) {
            // 同じ値がn回続いたらチャタリング影響なしとみなして値を確定
            if( same ) same--;
            if( !same ) {
                swport = swport1;
                scanon = 0;
            }
        }
        else {
            // not same
            same = CHATTER_CNT;
        }
    }
    else if( swport != swport1 ) {
        scanon = 1;
        same = CHATTER_CNT;
    }
    swport2 = swport1;      /* 最新のポートデータを保存 */

    return swport;
}

/*====================================
    REMOTE MODE TIMER COUNTER GET
  ====================================*/
unsigned short remote_timer_counter(unsigned char * direction,unsigned char * stop,unsigned char *highspeed)
{
    *direction = OFF;
    *stop = OFF;
    *highspeed = OFF;
    return 100000;
}

/*================================
    setup timer counter
  ================================*/
#define setup_tcnt(tcntset) {¥
    CCPR1L = LDATA(tcntset);¥
    CCPR1H = HDATA(tcntset);¥
}

/*====================================
    TIMER1 INTERRUPT ROUTINE
  ====================================*/
void timer1_isr_direct(void)
{
    static unsigned char phase = 0;
    static unsigned char outpulse = 0;

    /* OUTPUT PULSE(なるべく単純な処理で即座にパルスを出力する) */
    PORTA = outpulse;

    /* モーター出力位相更新 */
    phase += phase_inc;
    phase %= sizeof(pulse_tbl_12);
    outpulse = pulse_tbl[phase];
}

void timer1_isr_softtimer(void)
{
    softtimer -= tcnt;
    if( 0<softtimer ) {
        if( MAX_SHORT>softtimer ) {
            //最後のタイムアウト値設定
            tcnt = softtimer;
            setup_tcnt(tcnt);
        }
    }
    else {
        //ソフトタイマータイムアウト
        //タイマ値再設定
        softtimer = softtimer_set;
        tcnt = softtimer / softtimer_cnt;
        setup_tcnt(tcnt);

        //タイムアウト処理実行
        timer1_isr_direct();
    }
}

/*============================
    common interrupt routine
  ============================*/
void interrupt isr(void)
{
    /* CCP1 Interrupt ? */
    if( CCP1IF ) {
        CCP1IF = 0;     /* clear interrupt flag */

        timer1_isr();
    }
}

/*============================
    initialize
  ============================*/
unsigned char initialize(void)
{
    unsigned char readsw;
    unsigned char ii;
    unsigned char modesw;       // 0:STAR/1:KINGS/2:SUN/3:MOON/4:LANDSCAPE_STAR/5:DEC/6:COMET/7:OPTION

    /* initialize */
    di();
    PORTA = INIT_PORTA;
    PORTB = INIT_PORTB;
    CMCON = SET_CMCON;
    TRISA = SET_TRISA;
    RBPU = 0;               // PORTB internal pull-up enable
```

```
624:     TRISB = SET_TRISB;
625:     T1CON = SET_T1CON;
626:     ei();
627:
628:     __delay_ms(5);
629:     readsw = read_sw(-1);
630:     modesw = IN_MODE_SW(readsw);
631:     tcnt = timer_counter(modesw);
632:
633:     motor_type = E2promRead(EEP_MOTORTYPE);
634:     reiji_mode = E2promRead(EEP_REIJI);
635:
636:     /* EEPROMデータをRAMに読み込む */
637:     for(ii=0;ii<sizeof(eepdata);ii++)
638:     {
639:         eepdata[ii] = E2promRead(ii);
640:     }
641:
642:     di();
643:
644:     /* setup pulse phase table */
645:     pulse_tbl = pulse_ptbl[reiji_mode];
646:
647:     /* TIMER1 settings */
648:     TMR1CS = 0;             // Use internal 1/4 clock
649:     T1CKPS0 = 0;            // Prescaler 1/1
650:     T1CKPS1 = 0;
651:     T1SYNC = 0;             // Synchronize
652:     TMR1ON = 1;             // TIMER1 ON
653:     TMR1L = 0;
654:     TMR1H = 0;
655:
656:     /* CCP1 settings */
657:     CCP1CON = 0B00001011;   // Compare clear if equal
658:     setup_tcnt(tcnt);
659:
660:     CCP1IF = 0;             /* CCP1 interrupt flag clear */
661:     CCP1IE = 1;             /* CCP1 interrupt enable*/
662:     PEIE = 1;               /* peripheral interrupt enable */
663:     GIE = 1;                /* general interrupt enable */
664:
665:     ei();
666:
667:     return readsw;
668: }
669:
670: /**
671:  *  リモート状態判定処理
672:  */
673: unsigned char is_remote_mode(void)
674: {
675:     return remote_mode;
676: }
677:
678: /*======================
679:     setup mode
680:   ======================*/
681: #if defined(RARVSFUNC) && defined RVSSLOWCTL
682: #define ra_normal() {¥
683:                 tcntmode = mode;¥
684:                 if( s_delay ) {¥
685:                     s_delay = 0;¥
686:                     laststop = ON;¥
687:                 }¥
688: }
689: #elif defined(RARVSFUNC)
690: #define ra_normal() {
691:                 tcntmode = mode;¥
692:                 if( s_delay ) {¥
693:                     s_delay = 0;¥
694:                 }¥
695: }
696: #else
697: #define ra_normal() {¥
698:                 tcntmode = mode;¥
699: }
700: #endif
701: void modeset(unsigned char readsw,unsigned char mode)
702: {
703: #ifdef RVSSLOWCTL
704:     static unsigned char lastdirection = 0xff;  //前回の回転方向状態保持
705:     static unsigned char laststop = 0xff;       //前回の停止状態保持
706: #ifdef RARVSFUNC
707:     static unsigned char lastfast = 0xff;       //前回の高速指示状態保持
708: #endif
709: #endif
710:     static unsigned char lastmode = 0xff;
711:     unsigned char tcntmode = mode;
712:     unsigned char s_delay = stop_delay;
713:
714:     if( is_remote_mode() ) {
715:         /* REMOTE CONTROL MODE */
716:         tcnt = remote_timer_counter(&direction,&stop,&fast);
717:     }
718:     else {
719:         /* MECHANICAL CONTROL MODE */
720:
721:         /* PUSH SW OPERATION(LOGICAL SWITCH) */
722:         directionsw = IN_DIR_SW(readsw);
723:         stopsw = IN_STOP_SW(readsw);
724:         fastsw = IN_FAST_SW(readsw);
725:
726:         direction = directionsw;
727:
728:         if( MOTOR_DIRECT==motor_type ) {
729:             direction = !direction;
730:         }
731:
732:         if( stopsw && fastsw ) {
733:             stop = ON;
734:             fast = OFF;
735:         }
736:         else if( stopsw ) {
737:             stop = ON;
738:             fast = OFF;
739:         }
740:         else if( fastsw ) {
741:             stop = OFF;
742:             fast = ON;
743:         }
744:         else {
745:             stop = OFF;
746:             fast = OFF;
747:         }
748:
```

```
749:         if( MODE_DEC==mode || MODE_COMET==mode ) {
750:             /* DEC MODE(COMETモード時も挙動を同じにする) */
751:             if( stop ) {
752:                 //fastとは反対方向に動かす
753:                 direction = !direction;
754:             }
755:
756:             //どちらかのスイッチがONの時は必ず回転させる
757:             if( stop && softtimer_set ) {
758:                 //ソフトタイマー設定がされたということはコメットモードで移動量が少ない赤緯制御
759:                 tcntmode = MODE_DEC;
760:                 stop = OFF;
761:             }
762:             else if( fast ) {
763:                 //高速回転させる
764:                 tcntmode = MODE_DEC;
765:                 stop = OFF;
766:             }
767:             else if( stop ) {
768:                 if( MODE_DEC==mode ) {
769:                     //高速回転させる
770:                     tcntmode = MODE_DEC;
771:                 }
772: #ifdef RARVSFUNC
773:                 else {
774:                     //コメットモードでソフトタイマー未設定ということは赤経側
775:                     //星景モードの速度で逆転させる
776:                     tcntmode = MODE_LANDSCAPE_STAR;
777:                     s_delay = RARVSFUNC_DELAY_CNT;
778:                 }
779:                 stop = OFF;
780: #endif
781:             }
782:             else if( MODE_DEC==mode ){
783:                 //通常赤緯モードでどちらのスイッチも押されていない時は停止状態
784:                 stop = ON;
785:             }
786:             else {
787:                 //コメットモードでどちらのスイッチも押されていない時は定常状態
788:                 ra_normal();
789:             }
790:         }
791:         else if( MODE_OPTION==mode ) {
792:             //処理未定義
793:             tcntmode = mode;
794:         }
795:         else {
796:             /* RA MODE */
797:
798:             /* TIMER COUNT GET */
799:             if( fast ) {
800:                 tcntmode = MODE_DEC;
801:             }
802: #ifdef RARVSFUNC
803:             else if( stop ) {
804:                 //星景モードの速度で逆転させる
805:                 tcntmode = MODE_LANDSCAPE_STAR;
806:                 direction = !direction;
807:                 stop = OFF;
808:                 s_delay = RARVSFUNC_DELAY_CNT;
809:             }
810: #endif
811:             else {
812:                 ra_normal();
813:             }
814:         }
815:     }
816:
817: #if defined(RVSSLOWCTL) || defined(RARVSFUNC)
818:     if( stop==laststop && fast==lastfast && direction==lastdirection && mode==lastmode ) {
819:         //論理スイッチの状態が変っていないのでタイマカウンタ更新させない
820:         return;
821:     }
822:     stop_delay = s_delay;
823: #endif
824:
825:     tcnt = timer_counter(tcntmode);
826:     setup_tcnt(tcnt);
827:     if( 0==tcnt ) {
828:         stop = ON;
829:     }
830: #ifdef RVSSLOWCTL
831:     else if( !stop && stop==laststop && lastdirection!=direction ) {
832:         //回転パルス状態が前回と逆転状態になったので停止状態を挟んで遷移させる
833:         rvs_delay = RVS_DELAY_CNT;
834:     }
835:     lastdirection = direction;
836:     laststop = stop;
837: #endif
838: #ifdef RARVSFUNC
839:     lastfast = fast;
840: #endif
841:     lastmode = mode;
842: #ifdef DBGMON
843:     eepwrite(EEP_SW,readsw);
844:     eepwrite(EEP_MODESW,mode);
845:     if( MODE_OPTION!=mode ) {
846:         eepwrite_short(EEP_MONITOR_TIMER+(mode<<1),timer_counter(mode));
847:     }
848:     if( MODE_COMET==mode ) {
849:         eepwrite_long(EEP_MONITOR_SOFTTIMER,softtimer_set);
850:     }
851: #endif
852: }
853:
854: /*-----------------------------------
855:         option change
856:   -----------------------------------*/
857: void option_change(unsigned char readsw,unsigned char modesw)
858: {
859: #ifdef OPTION_CHANGE
860:     static unsigned char last_modesw = MODE_STAR;
861:     static unsigned char setcnt = 0;
862:     static unsigned char latch = 0;
863:     static unsigned short comet_tcnt;
864:     unsigned char stopsw = IN_STOP_SW(readsw);
865:     unsigned char fastsw = IN_FAST_SW(readsw);
866:
867:     if( comet_tcnt_set ) {
868:         /* カスタムモード、タイマカウンタ設定中 */
869:         if( OFF==latch && ON==stopsw ) {
870:             // 入力ラッチ
871:             // スピードスイッチの状態を1ビットずつ取り込む
872:             comet_tcnt <<= 1;
873:             comet_tcnt |= fastsw;
```

```
            setcnt++;
            if( setcnt>=16 ) {
                //カウンタ取り込み完了
                eepwrite_short(EEP_COMET_TIMER,comet_tcnt);
                comet_tcnt_set = OFF;
            }
        }
        latch = stopsw;
    }
    else if( modesw==last_modesw ) {
        /* モードの変化がなければ何もせずに抜ける */
        return;
    }
    else if( CHANGE_REIJI==modesw && stopsw ) {
        /* 励磁モード切替 */
        reiji_mode++;
        reiji_mode %= 2;
        eepwrite(EEP_REIJI,reiji_mode);

        /* setup pulse phase table */
        di();
        pulse_tbl = pulse_ptbl[reiji_mode];
        ei();
    }
    else if( CHANGE_MOTORTYPE==modesw && fastsw ) {
        /* モータ-タイプ切替 */
        motor_type++;
        motor_type %= NUM_MOTOR_TYPE;
        eepwrite(EEP_MOTORTYPE,motor_type);
    }
    else if( CHANGE_COMET==modesw && stopsw && fastsw ) {
        /* COMETモード時設定操作モード遷移 */
        comet_tcnt_set = ON;
        comet_tcnt = 0;
        setcnt = 0;
        latch = stopsw;
    }
    last_modesw = modesw;
#endif
}

/********************
    MAIN
 ********************/
void main(void)
{
    unsigned char last_sw = initialize();
    modeset(last_sw,IN_MODE_SW(last_sw));
    while(1)
    {
        /* sw data read */
        unsigned char readsw = read_sw(0);
        unsigned char modesw = IN_MODE_SW(readsw);

        if( readsw!=last_sw ) {
            option_change(readsw,modesw);
            modeset(readsw,modesw);
            last_sw = readsw;
        }
        // 初期化ディレイタイマカウント
        if( initphase ) {
            /* 初期起動してしばらくは設定値読み取りが安定していないのでパルスを更新出力しない */
            phase_inc = 0;
            initphase--;
        }
        else if( stop ) {
            /* stop状態の時もパルス更新出力しない */
            phase_inc = 0;
        }
#ifdef RVSSLOWCTL
        else if( rvs_delay ) {
            //逆転操作の時は一定時間パルス更新出力しない
            phase_inc = 0;
            rvs_delay--;
        }
#endif
#ifdef RARVSFUNC
        else if( stop_delay ) {
            //stop keyで逆転させる場合の遅延時間計時中はパルス出力を更新しない
            phase_inc = 0;
            stop_delay--;
        }
#endif
        else if ( comet_tcnt_set ) {
            /* COMETモードタイマカウンタ設定操作 */
            // 断続的にパルス出力を行う
            static unsigned char blink = 0;
            blink++;
            if( 0==blink ) {
                phase_inc++;
                phase_inc &= 1;
            }
        }
        else {
            /* パルスモータ位相更新設定 */
            if( direction ) {
                /* reverse */
                phase_inc = -1;
            }
            else {
                /* forword */
                phase_inc = 1;
            }
        }
    }
}
```