AN204780

F²MC-16FX Family, Stepper Motor Controller

This application note describes the functionality and usage of the SMCs.

Contents

1 Introduction

The MB96380 Series of MCUs features Stepper Motor Controllers (SMC). The function and usage of the SMCs is explained in this application note. For a full understanding, the functioning of a Stepper Motor is considered.

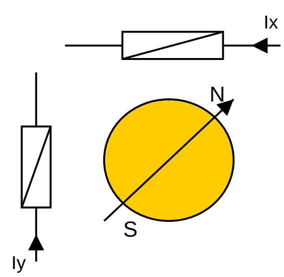
2 Stepper Motor

The SMC can easily be used for very smooth movement of a stepper motor as for example an indication application. Therefore the physical characteristics and properties have to be well known and understood.

2.1 Stepper Motor physics

For operation it might be useful to have a small introduction, what happens with the physics. In this explanation, we will use a simple replacement model for the stepper motor as a replacement circuitry. It represents the rotor as one bipolar magnet and two coils which are arranged perpendicular to each other as the stator (Figure 1).

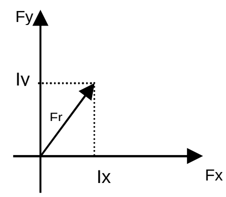
Figure 1. Replacement circuit for Stepper Motor



To meet the necessities of a really smooth movement, we have to insure that the torque moment is constant while the whole movement is performed. The preparation of this is done by adding the single coil components in a geometrical manner to a constant result (Figure 2).

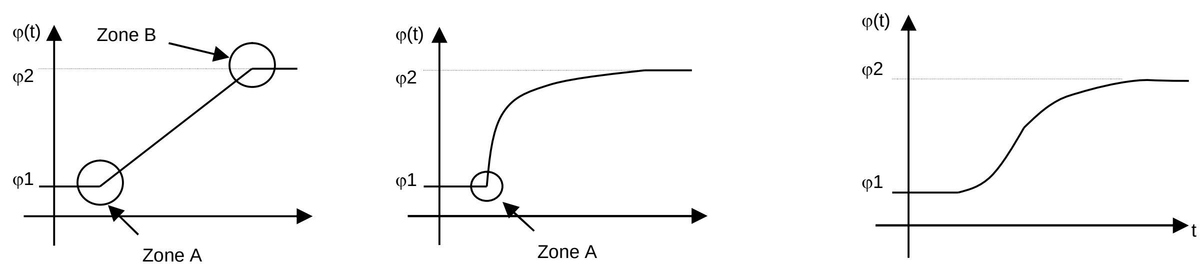
Normally to realize this, we use sine and cosine components for each coil. Therefore, with this model, we can position the rotor in each random position with the same torque moment.

Figure 2. Torque Moment calculation



For a movement, we have to perform a follow up of positions between a Start and a Stop position. While controlling usual stepper motors as real stepping devices, this is done with a step by step method (Figure 3). So, the movement has constant speed while moving. That is not suitable for smooth movement, e.g. if the motor arrives at the stop position it stops then abruptly.

Figure 3: Without filter, with 1st order LP filter, and with 2nd order LP filter



To improve this, a low-pass filter is used. The low-pass filter solves the problem at the stop point (Figure 3). Nevertheless, the same problem exists at the starting point. To solve this, we use a second order low-pass filter. The second order low-pass filter solves the problem at the start point (Figure 3), but it sets up the necessities of a maximum Speed and a max. Acceleration, which depends on the way of the movement.

Otherwise the motor itself has a physically given maximum speed and acceleration. To insure that the required properties do not overtake the given physics, we use an acceleration and velocity clipper. This clipper must be integrated into the 2nd order LP-filter, which must equalize the clipped edges.

The realization of a second order low-pass filter can be done by simply combining two stages of a 1st order LP-Filter. The 1st order LP-Filter can be performed by a simple mathematical formula like this:

$$PT1_i = \frac{Xin_i + (PT1_{i-1} * n)}{n+1} \qquad PT2_i = \frac{PT1_i + (PT2_{i-1} * n)}{n+1}$$

2.2 Control Theory

To implement the required 2nd order LP-filter in a way to get a fast calculation of it, it is useful to perform

1. Y1st_new= (((Y1st_old<<n) -Y1st_old) +Xin_new) >>n
2. Y2nd_new= (((Y2nd_old<<n) -Y2nd_old) +Y1st_new) >>n

This way, only two shift operations and two subtractions have to be performed. This saves CPU Duty cycles.

This operation has to be performed repetitively in a given time. The difference between the new output value at this time and the last output value at the last time is the velocity. So the actual speed of the requested movement can be evaluated by a simple subtraction.

If we store the actual speed in a memory cell, we can subtract the actual speed from the speed of the last time. The result is the actual acceleration. The physical units are as follows:

$$V_{act} = \frac{|PT2_i - PT2_{i-1}|}{t_2 - t_1} \quad \text{And} \quad a_{act} = \frac{|V_i - V_{i-1}|}{t_2 - t_1}$$

Or more suitable for programming:

phi_1 - phi_2 = d_phi

t1 - t2 = d_t

velo = d_phi / d_t

acc = (d_phi1 - d_phi2) / ((d_t) * (d_t)) ; same as dphi² / dt²

To clip them at a given moment, we have to check whether they reach the limits.

velo_new = Y2nd_new - Y2nd_old

For sample we compare them to given constants. If they are beyond the limit, we replace the new output value with substitutes from the physical evaluation.

```
velo_new = Y2nd_new - Y2nd_old
if  ( velo_new - velo_old ) > max.acceleration.constant then
        max.Velo.actual = velo_old + max.acc.constant
else
     max.velo.actual = max.velo.constant
endif
if   velo_new  > max.velo.actual  then
     Y2nd_new = Y2nd_old + max.velo.actual
Endif
```

By using these equations a few hundred times per second (repetitively time is only a few milliseconds), we will succeed to earn a smooth movement of our pointer in this application.

In practical use, the damping value n should be in the range of 3.6, because of the characteristics of the 2nd order LP-filter.

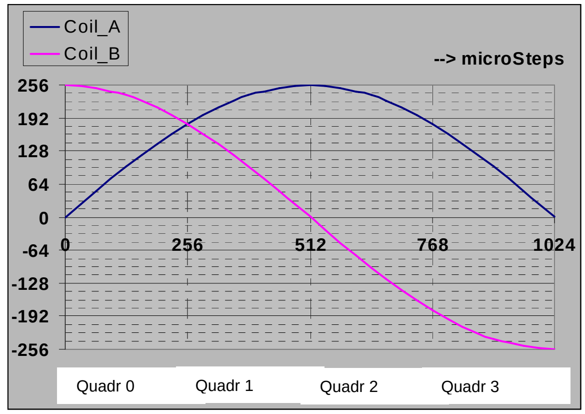
As an example, we can use lookup tables to cover the line switching at the output pins for the motor. Herewith a sample of an output functions for the stepper motor macro which uses a 128 micro steps per quadrant sine and cosine lookup table. Otherwise, the given pre-set value for this function is normalized to 256 micro steps per quadrant. Therefore, we can easily change The resolution for a given application from 0..7 Bits per quadrant, simply by changing the shift operation for normalizing and fetching the sine / cosine tables with the necessary length.

CPU_Pin : PWM2Mx → + Coil_B (- COS)

CPU_Pin : PWM2Px → + Coil_B (+ COS)

CPU_Pin : PWM1Px → + Coil_A (+ SIN)

CPU_Pin : PWM1Mx → - Coil_A (- SIN)



2.3 Code

Now we can realize this as a simple output function:

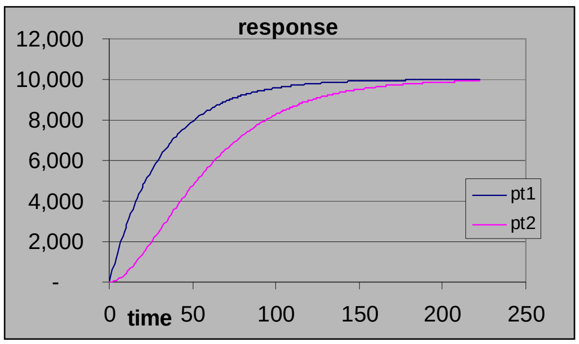
```
/* sin/cos Lookup table for microstepping */
unsigned char const SMC_TAB_CS[129]={
          0,  3,  6,  9, 13, 16, 19, 22,
         25, 28, 31, 34, 37, 41, 44, 47,
         50, 53, 56, 59, 62, 65, 68, 71,
         74, 77, 80, 83, 86, 89, 92, 95,
         98,100,103,106,109,112,115,117,
        120,123,126,128,131,134,136,139,
        142,144,147,149,152,154,157,159,
        162,164,167,169,171,174,176,178,
        180,183,185,187,189,191,193,195,
        197,199,201,203,205,207,208,210,
        212,214,215,217,219,220,222,223,
        225,226,228,229,231,232,233,234,
        236,237,238,239,240,241,242,243,
        244,245,246,247,247,248,249,249,
        250,251,251,252,252,253,253,253,
        254,254,254,255,255,255,255,255,
        255 };

/* Lookup tables for quadrant management */
unsigned char const smc_quad_a[4]={0x02, 0x10, 0x10, 0x02};
unsigned char const smc_quad_b[4]={0x50, 0x50, 0x42, 0x42};

void    smc_out(int ustp) {
        int q,d,smc_a,smc_b;    /* some squeeze intermediate memories      */
        q=((ustp>>8) & 3);      /* normalise the over all granulation
                                    to 1024 microsteps per polpair change   */
        d=((ustp>>1) & 127);    /* normalise the inner granulation
                                    to 512 microsteps per polpair change
                                    so that the Bit0 of ustp is don't care! */
        smc_a=SMC_TAB_CS[d];    /* preload of sin component                */
        smc_b=SMC_TAB_CS[128-d];/* preload of cos component
                                    note the trick with the enlarged table,
                                    which can be used in reverse order      */
        if ((q & 1)==1) {       /* decide where to go whatever             */
                PWC10=smc_a;    /* set up the sin value for coil A         */
                PWC20=smc_b;    /* set up the cos value for coil B         */
        }
        else {                  /* otherwise change the signs              */
                PWC10=smc_b;    /* set up the cos value for coil A         */
                PWC20=smc_a;    /* set up the sin value for coil B         */
        }
        PWC0=0xE8;              /* startover with the resource operation   */
        PWS10=smc_quad_a[q];    /* arming the signal for coil A            */
        PWS20=smc_quad_b[q];    /* arming the signal for coil B            */
}
```

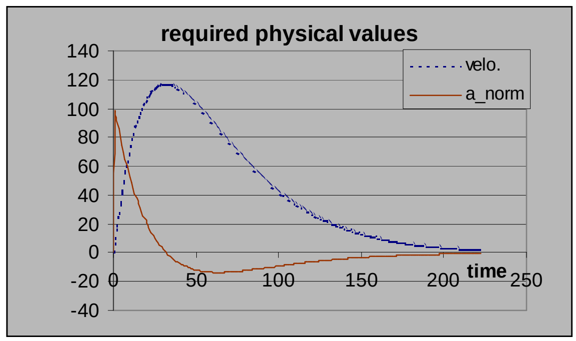
This output driver routine is hopefully a balanced finish between minimized memory requirements and clear viewing of coding effects.

Herewith an example of a 2nd order low-pass filter interrupts service routine to support the output function for the Stepper motor.

```
void    smc__lpf(void) {                    /* this tiny calculation should be done
                                            in a less part of a millisecond      */

        smc_old=smc_new;                    /* yesterdays future is passed today */

        /* first order low pass filter */
        *((int *)&smc_clc1+1)=smc_inp;      /* normalise input value             */
        smc_clc1=(smc_clc1>>smc_dn);
        smc_clc2=(smc_pt1-(smc_pt1>>smc_dn));
        smc_pt1=smc_clc2+smc_clc1;

        /* second order low pass filter */
        smc_clc2=(smc_pt2-(smc_pt2>>smc_dn));
        smc_pt2=smc_clc2+(smc_pt1>>smc_dn);

        smc_new=*((int *)&smc_pt2+1);       /* new output value                  */
}
```

This 2nd order Low pass filter will work in the following manner:





Therefore, we have to use a simple acceleration and velocity clipper to support the 2nd order low-pass filter.

```
void    smc_avclip(void) {                    /* limiting to the given physical values*/
        smc_clc1=(smc_new-smc_old);           /* actual velocity                     */
        if ( smc_clc1 < -smc_vmax  ) {        /* test for forward move               */
                 /* correction, because of velocity violation */
                smc_new=smc_old-smc_vmax;     /* set up new velocity                 */
                smc_clc1=-smc_vmax;           /* memorise new velocity               */
                *((int *)&smc_pt2+1)=smc_new; /* set up new output value             */
        }
        if ( smc_clc1 >  smc_vmax ) {         /* test for reverse move               */
                 /* correction, because of velocity violation                         */
                smc_new=smc_old+smc_vmax;     /* set up new velocity                 */
                smc_clc1=smc_vmax;            /* memorise new velocity               */
                *((int *)&smc_pt2+1)=smc_new; /* set up new output value             */
        }

        smc_acc=(smc_clc1-smc_velo);          /* actual acceleration                 */

        if ( smc_acc < -smc_amax ) {          /* test for acceleration               */
                 /* correction, because of acceleration violation                     */
                smc_clc1=smc_velo-smc_amax;   /* set up new velocity                 */
                smc_new=smc_old+smc_clc1;     /* recalculate output value            */
                *((int *)&smc_pt2+1)=smc_new; /* set up new output value             */
        }
        if ( smc_acc >  smc_amax ) {          /* test for deceleration               */
                 /* correction, because of acceleration violation                     */
                smc_clc1=smc_velo+smc_amax;   /* set up new velocity                 */
                smc_new=smc_old+smc_clc1;     /* recalculate output value            */
                *((int *)&smc_pt2+1)=smc_new; /* set up new output value             */
        }
        smc_acc =smc_clc1-smc_velo;           /* memorisation for debugging          */
        smc_velo=smc_clc1;                    /* memorisation for next cycle         */
}
```

This sample code is running in a special interrupt service routine, which should be called every few milliseconds.

```
__interrupt void irq_stepper_srv (void) { /* background task for motor controlling */
       smc_out(smc_new);    /* force the output first, for less delay glitch    */
       smc__lpf();          /* calculate next cycle output value                */
       smc_avclip();
       TMCSR1_UF = 0;       /* reset underflow interrupt request flag           */
}
```

Let us also assume that a potentiometer is connected to one of the ADC channel. The actual position of the rotor of potentiometer is converted into a digital data. These digital data are used as a controlling parameter for displacement of a pointer which is connected to stepper motor.

```
/*-------------------------------------------------------------------*/

void adc_init(void)
{
        DDR05_D0 = 0;
        ADER1     = 0x01;       /* P05_0                            */
        ADECR     = 0x00;
        ADCS_MD  = 0x01;        /* Select single mode 2             */
        ADCS_STS = 0x00;        /* Select software trigger          */
        ADSR_ST =  0x06;        /* Sampling time 3us @ 16Mhz        */
        ADSR_CT =  0x05;        /* Sampling time 8.3us @ 16Mhz      */
        return;
}

/*-------------------------------------------------------------------*/

int adc_sample(int chn)
{
        int     i,y;
        ADSR_ANS = chn;
        ADSR_ANE = chn;
        y=0;

        while ((ADCSH & 0x80) != 0);
        for (i=0; i<4; i++)
        {
                ADCSH=0x82;
                while ((ADCSH & 0x80) != 0);
                y=y+ADCR;
        }
        return (y);
}

/*-------------------------------------------------------------------*/

void smc_init(unsigned int x) /* set up all neccesary CPU resources  */
{
 /* initialise cpu output port for the motor */
        DDR10=0xFF;             /* assign pins as output              */
        PHDR10 = 0xFF;
        PWEC4 = 0x03;
 /* initialise low pass filters              */
        smc_inp=0;              /* clear target position    */
        smc_pt1=0; smc_pt2=0;  /* clear actual position              */
        smc_new=0; smc_old=0;  /* clear actual outputs               */
 /* initialise variables for physical limits */
        smc_dn= 4;              /* set up damping grade               */
        smc_vmax=100;           /* set up velocity limit              */
        smc_amax=  5;           /* set up acceleration limit          */
 /* initialise reload timer 1                  */
  TMRLR1  = x/2;                /* set reload value in [us]           */
  TMCSR1  = 0x81B;              /* prescaler 2us at 16MHz             */
}
```

```
#define PWC  PWC4
#define PWCA PWC14
#define PWCB PWC24
#define PWSA PWS14
#define PWSB PWS24


/*----------- Global Variables ------------------------------------*/

long    smc_pt1,  smc_pt2, smc_clc1, smc_clc2;
int     smc_inp,  smc_new,  smc_old, smc_velo,  smc_acc;

int     smc_dn,  smc_vmax, smc_amax;

unsigned int count;


/*-----------------------------------------------------------------*/
/* main program :                                                  */
/*-----------------------------------------------------------------*/

void main(void)              /* this is the MAIN program          */
{
long   tm, delay;            /* for support timing control        */
unsigned char i = 0;

  InitIrqLevels();           /* init IRQ Controller               */
  __set_il(7);               /* allow all levels                  */

  adc_init();                /* init Data acquirement             */
  smc_init(170);             /* Background support  x us          */
  DDR09 = 0xFF;

    for (smc_inp = 0xD000; smc_inp > 0; smc_inp--)
  {
    smc_out(smc_inp);
    for (delay=0; delay < 20; delay++)
      __asm("\tNOP");
  }

  __EI();                    /* global enable interrupts          */

  while(1)                   /* for ever                          */
  {

    for (tm=3000; tm>0; tm--);      /* wait time                  */
    smc_inp = adc_sample(8) * 6;
    PDR09 = ADCRL;                  /*Show some thing on LED      */
  }
}
```

Above example is prepared for SK-96380-120PMT board.

The appropriate output lines of the stepper motor controller are assigned to high power output stages, which have the capability to drive up to 30 mA, so that small stepper motors can be driven in direct manner, and bigger ones can be driven easily by connecting a power bridge to these pins.

3 Additional Information

Information about Cypress Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

4 Document History

Document Title: AN204780 - F²MC-16FX Family, Stepper Motor Controller

Document Number:002-04780

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	06/11/2006	Initial release, based on spectrum article of JMe
			09/24/2007	Updated Example
			09/28/2007	Updated Code formatting
*A	5080654	NOFL	04/26/2016	Migrated Spansion Application Note MCU-AN-300231-E-V12 to Cypress format

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Lighting & Power Control	cypress.com/powerpsoc
Memory	cypress.com/memory
PSoC	cypress.com/psoc
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless/RF	cypress.com/wireless

PSoC® Solutions

cypress.com/psoc

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Community | Forums |Blogs | Video |Training

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com