Fujitsu Microelectronics Europe
Application Note

MCU-AN-300077-E-V12

FR FAMILY

32-BIT MICROCONTROLLER

MB91460

STEPPER MOTOR CONTROLLER

APPLICATION NOTE

Revision History

Date	Issue
2008-02-26	V1.0, First draft, HPi
2008-03-20	V1.1, Updated example code, HPi
2008-06-17	V1.2 Chapter 4 Additional Information, sw example refernce corrected, MSt

This document contains 14 pages.

Warranty and Disclaimer

To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH restricts its warranties and its liability for **all products delivered free of charge** (eg. software include or header files, application examples, target boards, evaluation boards, engineering samples of IC's etc.), its performance and any consequential damages, on the use of the Product in accordance with (i) the terms of the License Agreement and the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials. In addition, to the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH disclaims all warranties and liabilities for the performance of the Product and any consequential damages in cases of unauthorised decompiling and/or reverse engineering and/or disassembling. **Note, all these products are intended and must only be used in an evaluation laboratory environment**.

1. Fujitsu Microelectronics Europe GmbH warrants that the Product will perform substantially in accordance with the accompanying written materials for a period of 90 days form the date of receipt by the customer. Concerning the hardware components of the Product, Fujitsu Microelectronics Europe GmbH warrants that the Product will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.

2. Should a Product turn out to be defect, Fujitsu Microelectronics Europe GmbH´s entire liability and the customer´s exclusive remedy shall be, at Fujitsu Microelectronics Europe GmbH´s sole discretion, either return of the purchase price and the license fee, or replacement of the Product or parts thereof, if the Product is returned to Fujitsu Microelectronics Europe GmbH in original packing and without further defects resulting from the customer´s use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to Fujitsu Microelectronics Europe GmbH, or abuse or misapplication attributable to the customer or any other third party not relating to Fujitsu Microelectronics Europe GmbH.

3. To the maximum extent permitted by applicable law Fujitsu Microelectronics Europe GmbH disclaims all other warranties, whether expressed or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the Product is not designated.

4. To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH´s and its suppliers´ liability is restricted to intention and gross negligence.

   NO LIABILITY FOR CONSEQUENTIAL DAMAGES

   To the maximum extent permitted by applicable law, in no event shall Fujitsu Microelectronics Europe GmbH and its suppliers be liable for any damages whatsoever (including but without limitation, consequential and/or indirect damages for personal injury, assets of substantial value, loss of profits, interruption of business operation, loss of information, or any other monetary or pecuniary loss) arising from the use of the Product.

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect

Contents

1 Introduction

The Stepper Motor Controller consists of PWM generators, motor drivers, selectors and A/D converter inputs. The ADC inputs can be used to measure the voltage level at the SMC pins during operation.

The SMCs have high current output drivers, where two motor coils can be connected directly (No further driver is necessary). The combination of the PWM generators and selector logic circuits is designed to control the motor rotation. The synchronization mechanism enables synchronous operation of two PWM generators.

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).

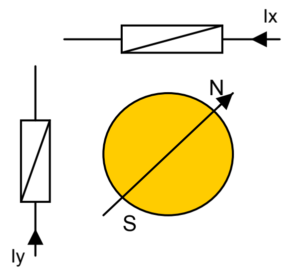


Figure 1: Replacement circuit for Stepper Motor

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 realise this, we use sine & cosine components for each coil. Therefore, with this model, we can position the rotor in each random position with the same torque moment.

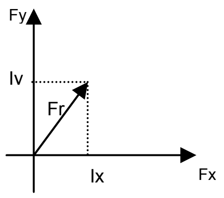


Figure 2: Torque Moment calculation

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.

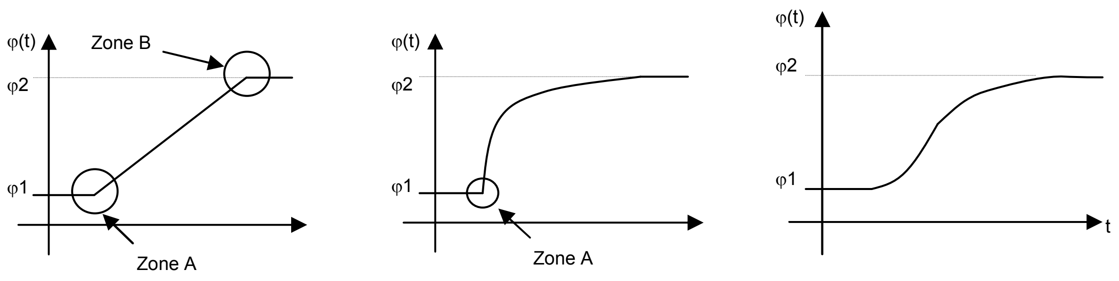


Figure 3: Without filter, with 1st order LP filter, and with 2nd order LP filter

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 equalise the clipped edges.

The realisation 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

i.) Y1st_new= (((Y1st_old<<n) -Y1st_old) +Xin_new) >>n

ii.) 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:

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

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
```

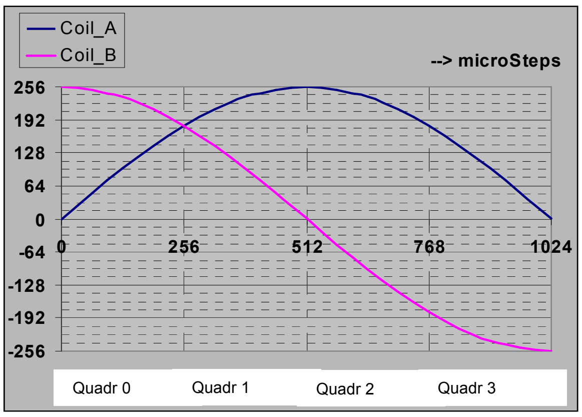
For sample we compare them to given constants. If they are beyond the limit, we replace the new output value with substitutes from the physical evaluation.

```
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 function for the stepper motor macro which uses a 128 microsteps per quadrant sine and cosine lookup table. Otherwise, the given pre-set value for this function is normalised to 256 microsteps 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 normalising 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 realise this as a simple output function:

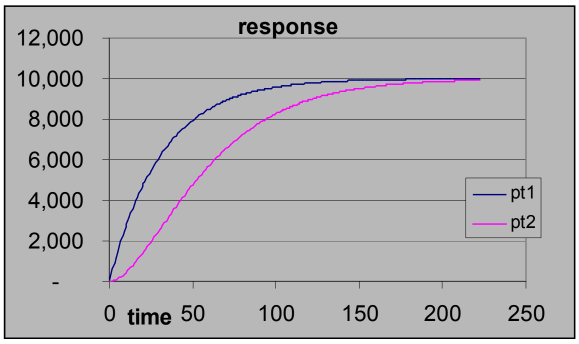
```
/* sin/cos Lookup table for microstepping */
unsigned char const SMC_TAB_CS[129]={
          0,  3,  6,  9, 13, 16, 19, 22,
         25, 28, 31, 34, 37, 41, 44, 47,
         50, 53, 56, 59, 62, 65, 68, 71,
         74, 77, 80, 83, 86, 89, 92, 95,
         98,100,103,106,109,112,115,117,
        120,123,126,128,131,134,136,139,
        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 */
}
```

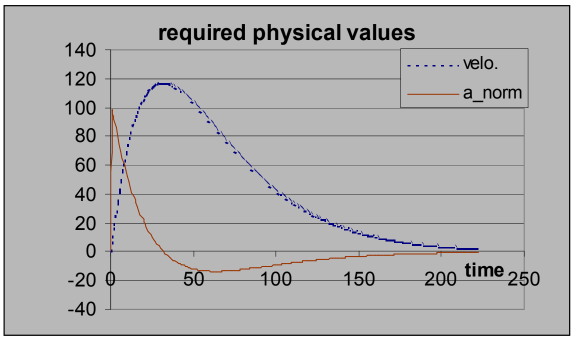
This output driver routine is hopefully a balanced finish between minimised memory requirements and clear viewing of coding effects.

Herewith an example of a 2nd order low-pass filter interrupt 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 Lowpass 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 ReloadTimer1 (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)
{
        PFR29    =  0xFF;         /*P29_00*/
        ADERL    =  0x0001;       /*AN0*/
        ADCS0    =  0x20;         /*MD1:0 = 00, S10 = 1*/
        ADCS1    =  0x0C;         /* STS1:0 = 11      */
        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++)
        {
                ADCS1 |= 0x02;
                while ((ADCS1 & 0x80) != 0);
                y=ADCR0;

                y = (207*y)+256;
        }
        return (y);
}

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

void smc_init(unsigned int x) /* set up all neccesary CPU resources */
{
 /* initialise cpu output port for the motor                  */
        PFR27=0x0F;              /* assign pins as output     */
/* 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 --------------------------------------*/

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

short 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;

  __set_il(31);              /* allow all levels */
  InitIrqLevels();           /* init interrupts */

  PORTEN = 0x3;
  adc_init();                 /* init Data acquirement                */
  smc_init(170);              /* Background support  x us             */

  DDR16 = 0xff;
  PFR16 = 0x00;

  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                             */
  {
    HWWD_CL = 0;
    for (tm=3000; tm>0; tm--);       /* wait time                     */
    smc_inp = adc_sample(0);
    PDR16 = ADCR0;                   /*Show some thing on LED*/
  }
}
```

Very small stepper motors can be driven in direct manner through MCU output pin, and bigger ones can be driven easily by connecting a power bridge to these pins.

3 Appendix

3.1 Figures

4 Additional Information

Information about FUJITSU Microcontrollers can be found on the following Internet page:
http://mcu.emea.fujitsu.com/

The software examples related to this application note is:
91460_smc2-vxx
It can be found on the following Internet page:
http://mcu.emea.fujitsu.com/mcu_product/mcu_all_software.htm