

F²MC-8L/16LX/FR FAMILY
8/16/32-BIT MICROCONTROLLER
ALL SERIES WITH SMC

**STEPPER MOTOR CONTROLLING
FOR POINTER APPLICATION**

APPLICATION NOTE

Revision History

Date	Issue
27/08/03	1.0 First draft, JMe / NFI

This document contains 17 pages.

Abbreviations:

FME	Fujitsu Microelectronics Europe GmbH
MCU	Microcontroller
SMC	Steppermotor Controller

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 from 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 inclusive 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

REVISION HISTORY	2
WARRANTY AND DISCLAIMER	3
CONTENTS	4
0 INTRODUCTION	5
1 THE PHYSICS	6
1.1 Performing a steppermotor controlling for a pointer application.	6
2 STEPPER MOTOR CONTROLLER	9
2.1 Microcontroller series with Steppermotor Controller	9
2.2 Steppermotor Controller Block	9
2.3 Steppermotor Controller Registers	10
2.3.1 PWM Control Register	10
2.3.2 PWM Compare Register	12
2.3.3 PWM Select Register	13
3 SOURCE CODES	15
3.1 Lookup table for microstepping	15
3.2 Low-pass filter	16
3.3 Sample of a interrupt service routine	17
3.4 Limiting to the given physical values	17

0 Introduction

Stepper motors are widely used in printers, automated machine tools, disk drives, automotive dashboard instrument clusters, and other applications requiring precise motions using computer control.

To simplify the design effort and reduce the cost of end products that use stepper motors, Fujitsu offers low-cost 8-, 16-, and 32-bit microcontrollers with integrated stepper motor drive circuits.

Special logic and high-current drive circuits are required to drive stepper motors. These can be designed using discrete logic or special interface ICs, which may result in either increased design complexity or increased end product cost, or both.

A common use for stepper motors is in automotive dashboard instrument clusters. Stepper motors are used to power the needles or pointers that indicate parameters, such as vehicle speed or the RPM of the engine. One or more stepper-motor controllers on Fujitsu's Flash Microcontroller can be individually programmed to control the speed gauge, the tachometer, the fuel gauge, and the engine temperature gauge.

This application note describes how to control a stepper motor by Fujitsu's Flash Microcontroller with SMC-driver for a pointer application.

1 The Physics

This chapter reflects the technical background of stepper motor controlling

1.1 Performing a stepper motor controlling for a pointer application.

Some of Fujitsu's MCUs carry a stepper motor controller macro resource. This can easily be used for very smooth movement of a stepper motor as for example as an indication application. Therefore the physical characteristics and properties have to be known really.

For operation it may be useful to have a small introduction, what's happening with the physics.

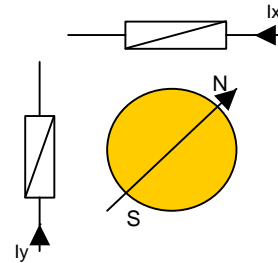
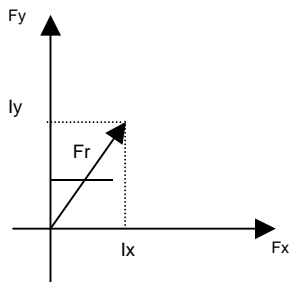


Figure 1: Function principle of stepper motor

In this explanation, we will have 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 rectangular to each other as the stator (Figure 1).



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 preparing of this is done by adding the single coil components in a geometrical manner to a constant result (Fig. 2).

To normalise realise this; we use sin & cos 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 of stepper motor

For a movement, we have to perform a follow up of positions between a Start and a Stop position. While controlling usual stepper motor as real stepping devices, this is done with a step-by-step method (Fig. 3).

So, the movement has constant speed while moving. That is not suited for smart movement, because, if the motor arrives at the stop position it stops than immediately.

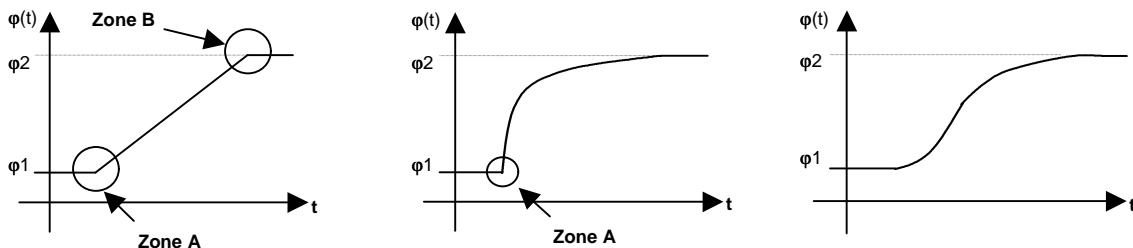


Figure 3: Low-pass filter

To improve this, we use a low-pass filter. The low-pass filter solves the problem at the stop point (Zone B).

Nevertheless, the same problem exists at the starting point. To solve this, we use a second order low-pass filter.

The 2nd order low-pass filter solves the problem at the start point (Zone A), 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 maximal 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 simple two stage using of a 1st order LP-Filter.

The 1st order LP-Filter can be performed, by a simple mathematical formula like this (Fig. 4).

$$PT1_i = \frac{Xin_i + (PT1_{i-1} * n)}{n + 1} \quad \text{and} \quad PT2_i = \frac{PT1_i + (PT2_{i-1} * n)}{n + 1} \quad [1]$$

Figure 4: Formula of Low-pass filter

To implement it in a way to become a fast calculation of it, so it is useful to perform

$$Y1st_new = (((Y1st_old << n) - Y1st_old) + Xin_new) >> n \quad [2]$$

$$Y2nd_new = (((Y2nd_old << n) - Y2nd_old) + Y1st_new) >> n \quad [3]$$

So only two shift operations and two subtractions have to be performed. This saved CPU Duty cycles.

This operation has to be performed in a given repetitively time. The difference between the new output value at this time and the last output value at the last time is the velocity. So that 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 by the last time speed. The result is the actual acceleration. The physically units are as follows:

$$V_{act} = \frac{|PT2_i - PT2_{i-1}|}{t_2 - t_1} \quad \text{and} \quad a_{act} = \frac{|V_i - V_{i-1}|}{t_2 - t_1} \quad [4]$$

Or programming slang spoken:

$$\begin{aligned} \text{phi}_1 - \text{phi}_2 &= d_phi & \text{there is: } t_1 - t_2 &= d_t \\ \text{velo} &= d_phi / d_t \\ \text{acc} &= (d_phi1 - d_phi2) / ((d_t) * (d_t)) ; \text{ same as } d\text{phi}^2 / dt^2 \end{aligned}$$

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

$$\text{velo_new} = Y2\text{nd_new} - Y2\text{nd_old}$$

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

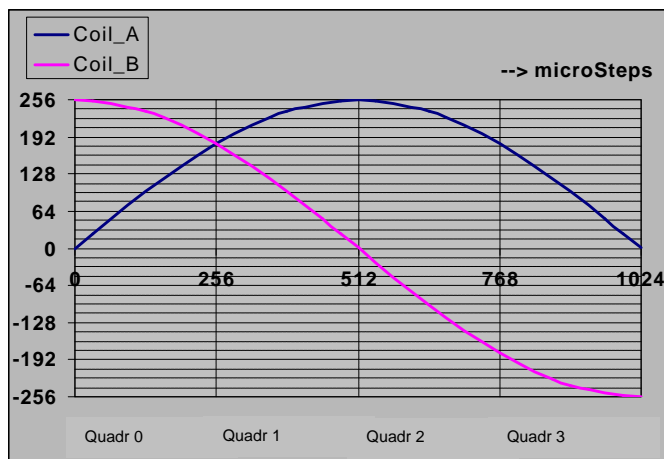
```

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

In practical use, the damping value **n** should in range of **3-6**, because of the characteristics of the 2nd order Low-pass filter.

For sample, we can use lookup tables to cover the line switching at the output pins for the Steppermotor.

Herewith a sample of an output function for the Steppermotor macro that uses a 128 micro-step per quadrant sinus and cosine lookup table. Otherwise, the given pre-set value for this function is normalised to 256 micro-steps per quadrant. Therefore, we can easily change the resolution for a given application from 0..7 Bits per quadrant, only by changing the shift operation for normalising and fetching the sinus / cosine tables with the necessity length.



```

CPU_Pin : PWM1Px → + Coil_A ( + SIN )
CPU_Pin : PWM1Mx → - Coil_A ( - SIN )
CPU_Pin : PWM2Px → + Coil_B ( + COS )
CPU_Pin : PWM2Mx → - Coil_B ( - COS )

```

Figure 5: Output function for the Steppermotor macro

2 Steppermotor Controller

This chapter shows the features of Steppermotor Controller

2.1 Microcontroller series with Steppermotor Controller

Fujitsu Microelectronics provides some microcontrollers with integrated SMC-drivers:

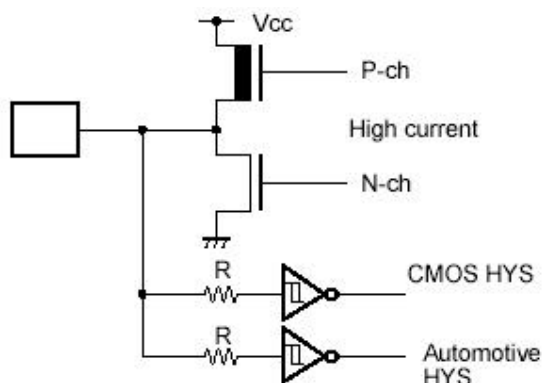
16LX MCU series	Type	SMC channels
MB90F394	16bit	6ch
MB90F427	16bit	4ch
MB90F428	16bit	4ch
MB90F591	16bit	4ch
MB90F594	16bit	4ch
MB90F598	16bit	4ch

FR MCU series	Type	SMC channels
MB91F362	32bit	4ch
MB91F365	32bit	4ch
MB91F366	32bit	4ch
MB91F368	32bit	4ch
MB91F376	32bit	4ch

8L MCU series	Type	SMC channels
MB89943	8bit	1ch
MB89945	8bit	1ch

2.2 Steppermotor Controller Block

The mentioned MCU series will be used exemplary to reflect the internal Steppermotor controller. The Steppermotor controller consists of four motor drivers, the corresponding selector logic, and two PWM pulse generators.



The four motor drivers have high-current drive capabilities capability to drive up to 30mA, and they can be directly connected to the four ends of two motor coils.

So, that very small steppermotor can be driven in direct manner, and bigger ones can be driven by easily connecting a power bridge to these pins.

Figure 6: Steppermotor output driver

The combination of the PWM Pulse Generators and Selector Logic is designed to control the rotation of the motor.

The Stepper motor controller block is divided into 2 channels to connect the four ends of two motor coils as described in the following illustration.

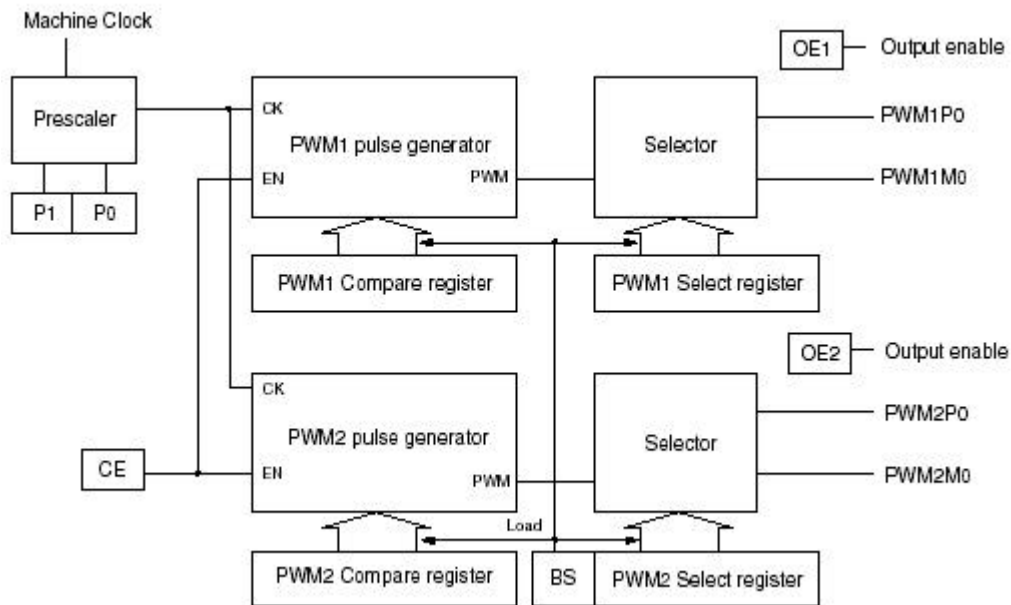


Figure 7: Block diagram of Stepper Motor Controller

2.3 Stepper Motor Controller Registers

The stepping motor controller "n" has the following five types of registers:

- PWM control n register (PWMCn)
- PWM1 compare n register (PWC1n)
- PWM2 compare n register (PWC2n)
- PWM1 select register (PWS1n)
- PWM2 select register (PWS2n)

2.3.1 PWM Control Register

The PWM control register (PWMCn) starts and stops the Stepper Motor Controller, controls the interrupts, and sets the external output pins. Its function is equal to all other SMC modules.

STEPPER MOTOR CONTROLLING
for Pointer Application
Chapter 2 Stepper Motor Controller

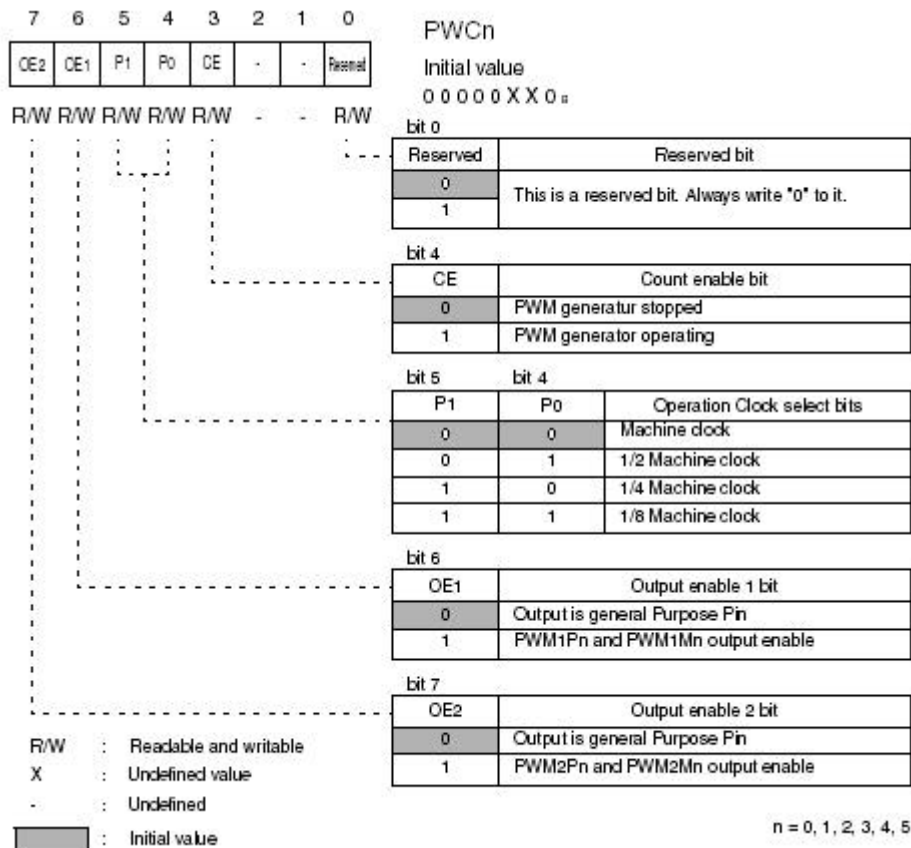


Figure 8: PWM control register

Bit name		Function															
bit 7	OE2: Output enable 2 bit	When this bit is set to "1", the external pins are assigned as PWM2Pn and PWM2Mn outputs. Otherwise they can be used as general purpose IO.															
bit 6	OE1: Output enable 1 bit	When this bit is set to "1", the external pins are assigned as PWM1Pn and PWM1Mn outputs. Otherwise they can be used as general purpose IO.															
bits 5, 4	P1, 0: Operation Clock select bits	These bits specify the clock input signal for the PWM pulse generators. <table border="1" style="margin: 5px auto;"> <tr><th>P1</th><th>P0</th><th>Clock Input</th></tr> <tr><td>0</td><td>0</td><td>Machine clock</td></tr> <tr><td>0</td><td>1</td><td>1/2 Machine clock</td></tr> <tr><td>1</td><td>0</td><td>1/4 Machine clock</td></tr> <tr><td>1</td><td>1</td><td>1/8 Machine clock</td></tr> </table>	P1	P0	Clock Input	0	0	Machine clock	0	1	1/2 Machine clock	1	0	1/4 Machine clock	1	1	1/8 Machine clock
P1	P0	Clock Input															
0	0	Machine clock															
0	1	1/2 Machine clock															
1	0	1/4 Machine clock															
1	1	1/8 Machine clock															
bit 3	CE: Count enable bit	This bit enables the operation of the PWM pulse generators. When it is set to "1", the PWM pulse generators start their operation. Note that the PWM2 pulse generator starts the operation one machine clock cycle after the PWM1 pulse generators is started. This is to help reduce the switching noise from the output drivers.															
bit 2, 1	Undefined																
bit 0	reserved bit	This is a reserved bit. Always write "0" to this bit.															

Figure 9: Function of each bit of the PWM control register

2.3.2 PWM Compare Register

The PWM1 and 2 compare registers (PWC1n + PWC2n) determine the widths of PWM pulses. The stored value of "00h" represents the PWM duty of 0% and "FFh" represents the duty of 99.6%. The two 8-bit compare registers are accessible at any time, however the modified values are reflected to the pulse width at the end of the current PWM cycle after the BS bit of the PWM2 Select register is set to "1".

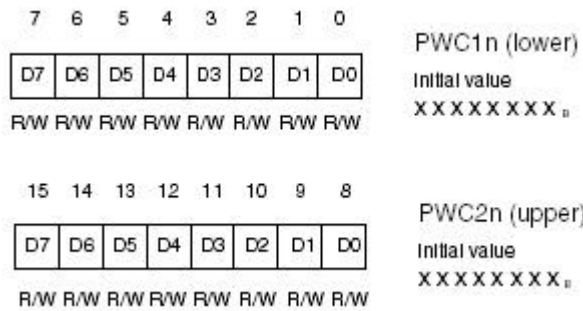


Figure 10: PWM1&2 compare registers

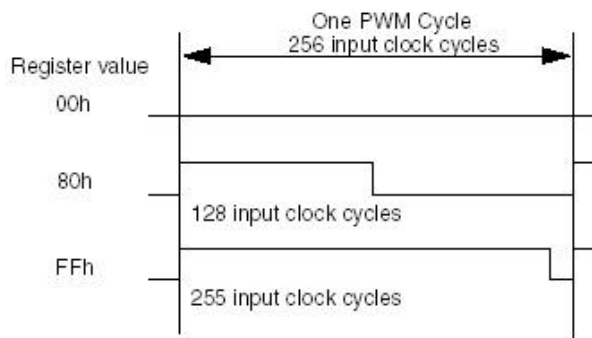


Figure 11: Examples for duty cycle settings

2.3.3 PWM Select Register

The PWM1 and PWM2 select registers (PWS1n + PWS2n) can choose between static low, static high, PWM pulse, or high impedance for the external output pin of the Steppermotor Controller.

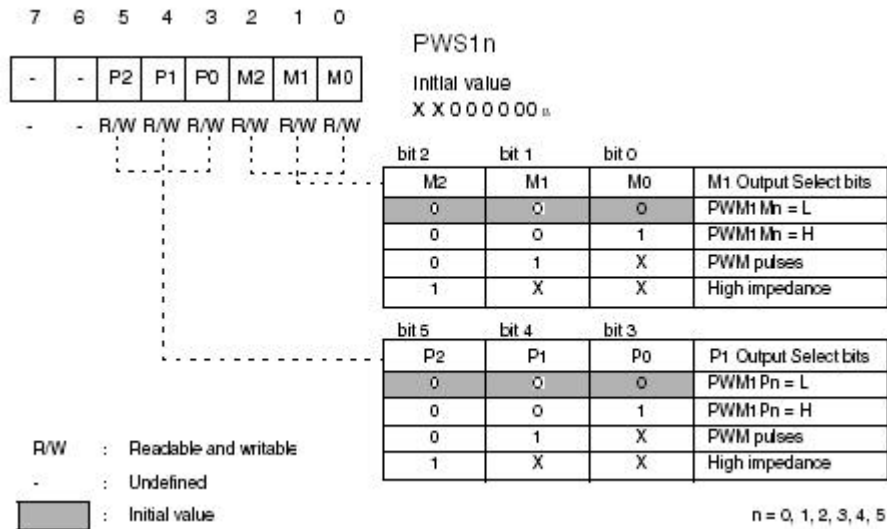


Figure 12: PWM1 select register

Bit name		Function																				
bit 7, 6	Undefined																					
bit 5 to 3	P2 to P0: Output P select bits	These bits selects the output signal at PWM1Pn <table border="1" style="margin: 10px auto;"> <thead> <tr> <th>P2</th> <th>P1</th> <th>P0</th> <th>PWM1Pn</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>L</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>H</td> </tr> <tr> <td>0</td> <td>1</td> <td>x</td> <td>PWM pulses</td> </tr> <tr> <td>1</td> <td>x</td> <td>x</td> <td>High impedance</td> </tr> </tbody> </table>	P2	P1	P0	PWM1Pn	0	0	0	L	0	0	1	H	0	1	x	PWM pulses	1	x	x	High impedance
P2	P1	P0	PWM1Pn																			
0	0	0	L																			
0	0	1	H																			
0	1	x	PWM pulses																			
1	x	x	High impedance																			
bit 2 to 0	M2 to M0: Output M select bits	These bits selects the output signal at PWM1Mn <table border="1" style="margin: 10px auto;"> <thead> <tr> <th>M2</th> <th>M1</th> <th>M0</th> <th>PWM1Mn</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>L</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>H</td> </tr> <tr> <td>0</td> <td>1</td> <td>x</td> <td>PWM pulses</td> </tr> <tr> <td>1</td> <td>x</td> <td>x</td> <td>High impedance</td> </tr> </tbody> </table>	M2	M1	M0	PWM1Mn	0	0	0	L	0	0	1	H	0	1	x	PWM pulses	1	x	x	High impedance
M2	M1	M0	PWM1Mn																			
0	0	0	L																			
0	0	1	H																			
0	1	x	PWM pulses																			
1	x	x	High impedance																			

Figure 13: Function of each bit of the PWM1 select register

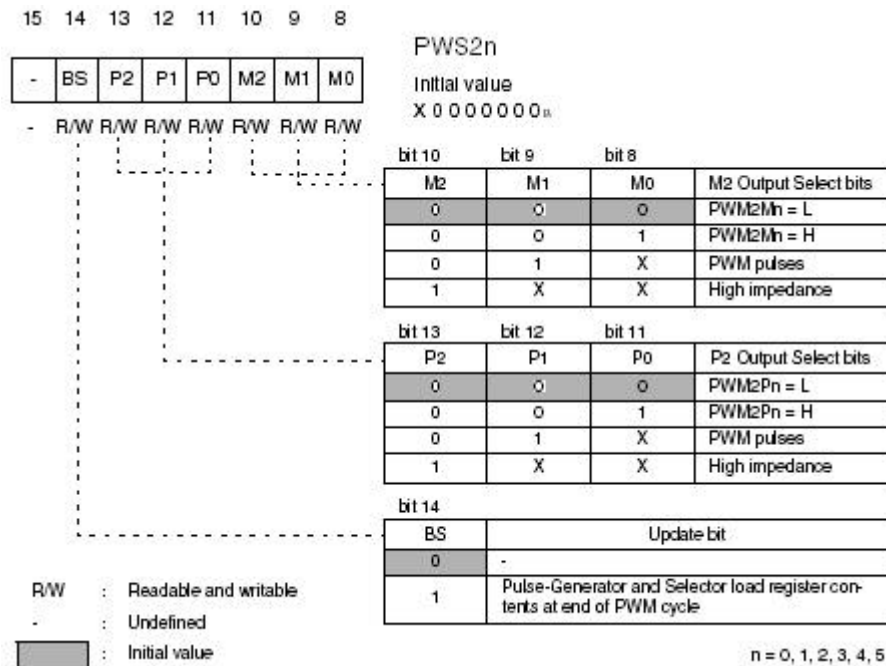


Figure 14: PWM2 select register

Bit name	Function																				
Bit 15	Undefined																				
bit 14	<p>BS: Update bit</p> <p>This bit is prepared to synchronize the settings for the PWM outputs. Any modifications in the two compare registers and two select registers are not reflected to the output signals until this bit is set.</p> <p>When this bit is set to "1", the PWM pulse generators and selectors load the register contents at the end of the current PWM cycle. The BS bit is reset to "0" automatically at the beginning of the next PWM cycle. If the BS bit is set to "1" by software at the same time as this automatic reset, the BS bit is set to "1" (or remains unchanged) and the automatic reset is cancelled.</p>																				
bit 13 to 11	<p>P2 to P0: Output P select bits</p> <p>These bits selects the output signal at PWM2Pn</p> <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>P2</th> <th>P1</th> <th>P0</th> <th>PWM2Pn</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>L</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>H</td> </tr> <tr> <td>0</td> <td>1</td> <td>x</td> <td>PWM pulses</td> </tr> <tr> <td>1</td> <td>x</td> <td>x</td> <td>High impedance</td> </tr> </tbody> </table>	P2	P1	P0	PWM2Pn	0	0	0	L	0	0	1	H	0	1	x	PWM pulses	1	x	x	High impedance
P2	P1	P0	PWM2Pn																		
0	0	0	L																		
0	0	1	H																		
0	1	x	PWM pulses																		
1	x	x	High impedance																		
bit 10 to 8	<p>M2 to M0: Output M select bits</p> <p>These bits selects the output signal at PWM2Mn</p> <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>M2</th> <th>M1</th> <th>M0</th> <th>PWM2Mn</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>L</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>H</td> </tr> <tr> <td>0</td> <td>1</td> <td>x</td> <td>PWM pulses</td> </tr> <tr> <td>1</td> <td>x</td> <td>x</td> <td>High impedance</td> </tr> </tbody> </table>	M2	M1	M0	PWM2Mn	0	0	0	L	0	0	1	H	0	1	x	PWM pulses	1	x	x	High impedance
M2	M1	M0	PWM2Mn																		
0	0	0	L																		
0	0	1	H																		
0	1	x	PWM pulses																		
1	x	x	High impedance																		

Figure 15: Function of each bit of the PWM2 select register

3 SOURCE CODES

This Chapter shows and explains the source code for a pointer application

3.1 Lookup table for microstepping

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics Europe GmbH */

/* 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 */
}
```

3.2 Low-pass filter

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

Herewith a sample of a 2nd order Low-pass filter interrupt to support the output function for the Steppermotor.

```

/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics Europe GmbH */

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 */

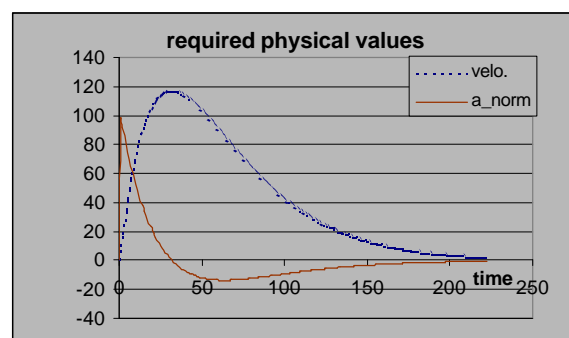
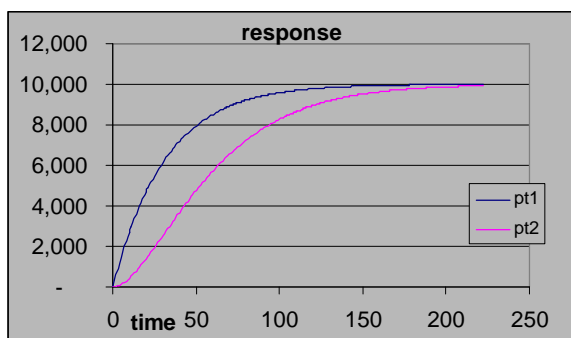
    /* first order low pass filter */
    *((int *)&smc_clc1+1)=smc_inp; /* normalise input value */
    smc_clc1=(smc_clc1>>smc_dn); /* */
    smc_clc2=(smc_pt1-(smc_pt1>>smc_dn));
    smc_pt1=smc_clc2+smc_clc1;

    /* 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 thru the following manner:



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

3.3 Sample of a interrupt service routine

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

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics Europe GmbH */

__interrupt void irq_stepper_srv (void) { /* background task for motor controlling
*/
smc_out(smc_new); /* force the output first, for less delay glitch */
  smc_ido(); /* calculate next cycle output value */
  smc_avclip(); /* */
  TMC SR1_UF = 0; /* reset underflow interrupt request flag */
}
```

3.4 Limiting to the given physical values

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics Europe GmbH */

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 reward 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 breaking */
    /* 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 */
}
```