CH-1092 Belmont, Switzerland
info@didel.com, www.didel.com

0.97

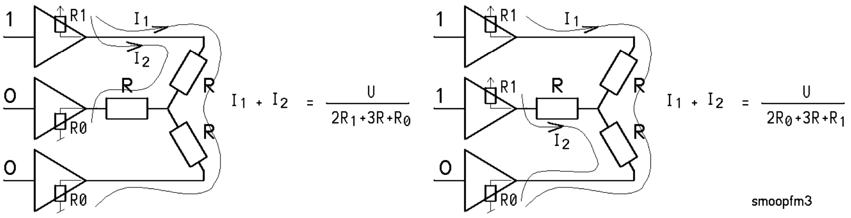
Open-loop control for smoovy motors

RMB smoovy motors are synchronous motors which can be controlled open-loop with a better efficiency using the PFM scheme proposed by O. Matthey. The basic idea is to smoothen the transitions in order to achieve close to the ideal sinusoid excitation for which synchronous motors are designed. The additional difficulty with miniature smoovy motors is that rotor inertia is very low compared to the magnetic forces. Careful experiment has shown that, with the PFM software constraints, trapezoidal transitions perform better. PFM is less time-consuming than PWM and is easier to implement on simple microcontrollers like those of the Microchip PIC family.

1. Interfacing

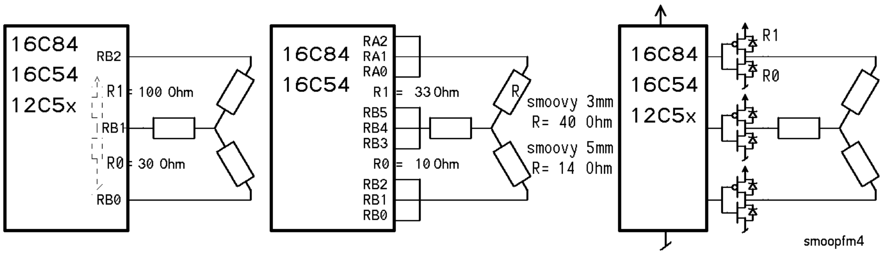
The smoovy, like all three-phase synchronous motors, has three coils around a rotor which is just the best possible magnet. The rotating electromagnetic field drive? the rotor with a phase shift that generate the active torque. If one can control this phase shift, as brushless motors with Hall sensors and analog electronics do, one must overpower the motor to be sure not to loose steps, and have the motor stop. Open-loop control will, however, always be the only way to go with the smallest motors.

The coil resistance of the 3mm is about 40 Ohm, which is a major advantage since it can be directly powered by some microcontrollers. Smaller motors do not have such a high resistance. The more powerful 5 mm smoovy has a 14 Ohm coil resistance. Power amplifiers have a resistance toward the supply or the ground which will define the efficiency of the system. Coils are never controlled individually. They are connected in a star configuration, with a common point which may provide an indication on the current. Triangle connections are seldom used.



smoopfm3 *Fig. 1 Star connection of the motor coils*

The amplifiers are easy to build with miniature low resistance MOS transistors (figure 2). Amplifiers are not required for the 3mm smoovy, if a reduced torque can be accepted. Connecting outputs together reduces the internal equivalent PIC resistance and is of course preferable, but the outputs for a given coil must be on the same port.



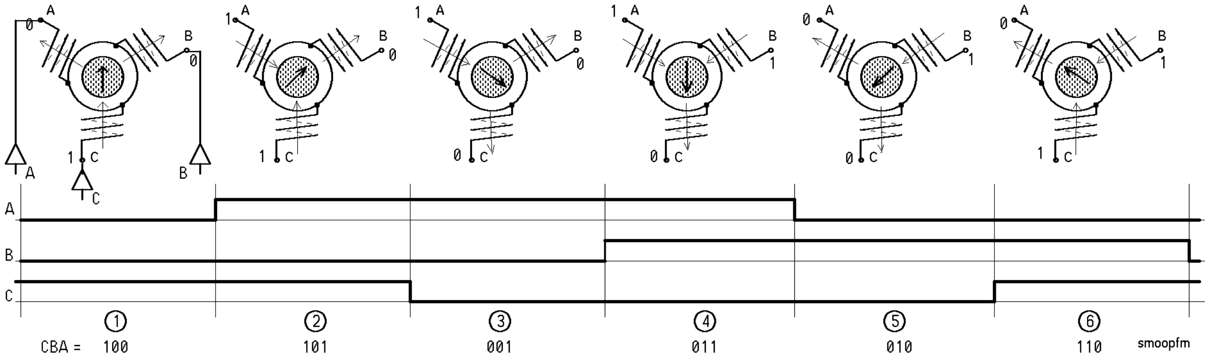
Miniature transistors		
Si3441DV Si3442DV	R1= 0.1 Ohm R0= 0.07 Ohm	TSOP-6
TP0101T TN0200T	R1= 0.45 Ohm R0= 0.29 Ohm	SOT-23
Miniature half bridges		
Si9952	R1= 0.4 Ohm R0= 0.1 Ohm	SO-8
Si4459EY	R1= 0.12 Ohm R0= 0.055 Ohm	SO-8
NDC7001C	R1= 5 Ohm R2= 2 Ohm	SOT-6

smoopfm4 *Fig. 2 Power amplifiers and typical resistance values*

2. Fixed frequency smoovy control

A table defines the sequence of steps (figure 3). If the PIC has no other task to perform, a simple delay loop defines the period between pulses, that is the rotation speed. Due to its low inertia, the smoovy will start at rather high frequency (about 1000 RPM). But for some lower speed, it may overshoot and not work correctly.

Half-steps are possible if the power amplifiers have an "enable" input. The processor must in this case generate 6 signals, and there are 12 steps. The torque is less regular, however, due to a lower total current when a coil is disconnected during a half-step. We will no longer consider half-steps.



smoopfm5

Fig. 3 Step sequence for a 3-phase motor

The software corresponding to figure 3 is quite simple. In the example below, the 3 phases are connected to the three low bits of PortB. A pointer onto the step sequence table is initialized at zero, but is incremented before accessing the data. Hence, the first position in the table is never accessed. At each step, the next value in the table is taken. When

Microchip source program

```
      LIST      P=16C84

C1       EQU    0xC
C2       EQU    0xD
MOTPOS EQU      0xE


;PORTB  16C84
BM1      EQU    0
BM2      EQU    1
BM3      EQU    2
MDIRB    EQU    0B0000000

; PROGRAM

BEGIN
      MOVLW     MDIRB
      TRIS      PORTB

PER   EQU       100

LOOP  CLRF      MOTPOS
L     MOVLW     PER
      CALL      DELAY
      MOVF      MOTPOS,W
      INCF      MOTPOS
      CALL      TAFORWARD
      MOVWF     PORTB
      MOVLW     6
      SUBWF     MOTPOS,W
      BTFFS     STATUS,3
      GOTO      L
      GOTO      LOOP



DELAYMOVWF      C1
A     MOVLW     32
      MOVWF     C2
B     DECFSZ    C2
      GOTO      B
      DECFSZ    C1
      GOTO      A
      RETURN


TAFORWARD
      ADDWF     2
      RETLW 0B100
      RETLW 0B110
      RETLW 0B010
      RETLW 0B011
      RETLW 0B001
      RETLW 0B101
```

CALM source program with SmileNG editor– Test smoovy

```
.proc      16c84     ; 4MHz clock

Variables Registers  .Loc       16'C
C1:        .16       1          ; local variables (counters, e
C2:        .16       1
MotPos:    .16       1          ; used by motor loop (motor

Variables PortB  16C84
bM1        = 0       ; RB0 Pin 7
bM2        = 1       ; Pin 8
bM3        = 2       ; Pin 9
mDirB      = 2'0000000          ; all outputs

Program Program
.Loc       0
Begin:
           Move      #mDirB,W ; Direction out
           Move      W,TrisB

Per        = 100     ; 10ms --> 60ms/turn 16,6t/s 1000t
           ; min 20 for unloaded motor
Loop:      Clr       MotPos     ; motor position index
 M$:       Move      #Per,W
           Call      Delay
           Move      MotPos,W
           Inc       MotPos
           Call      TaForward
           Move      W,PortB
           Move      #6,W       ; 6 phases per turn
           Sub       W,MotPos,W  ; Compare #6,MotPos
           Skip,EQ
           Jump      M$
           Jump      Loop

Routine Delay  Delay multiple of 100µs (4MHz clock)
  in:   W delay 0, 0,1 ... 25,5 ms
  mod: C1 C2 W
Delay:     Move      W,C1
 A$:       Move      #32,W      ; loop 100µs
           Move      W,C2
    B$:    DecSkip,EQ C2
           Jump      B$
           DecSkip,EQ C1
           Jump      A$
           Ret

Constant Tables  Motor
TaForward:                      ; Motor phases on bM3 bM2 b
           Add       W,PCL
           RetMove   #2'100,W           ; MotPos = 0
           RetMove   #2'110,W
           RetMove   #2'010,W
           RetMove   #2'011,W
           RetMove   #2'001,W
           RetMove   #2'101,W           ; MotPos = 5
```

the pointer has reached position 6, it is reinitialized to zero.

3. Reorganizing the table

Instead of a table organized by consecutive states, a table giving the next excitation according to the present one is more efficient, since there is no modulo-6 counter to manage. The variable which corresponds to the excitation status of the motor is "Excit". It could even be read on the motor port itself, but since the PIC reads the output value and not the internal output register, it is not recommended, especially in the case of capacitive loads. The "Excit" variable is initialized at zero, and the next valid value is found automatically in the table.

```
BEG
        CLR         EXCIT
LOOP
        MOVF        EXCIT,W
        CALL        TAFORWARD
        MOVWF       EXCIT

        MOVWF       6
        GOTO        M_0



TAFORWARD
        ANDLW       0b111
        ADDWF       2
        RETLW       0b001
        RETLW       0b101
        RETLW       0b011
        RETLW       0b001
        RETLW       0b110
        RETLW       0b100
        RETLW       0b010
```

Unidirectional control

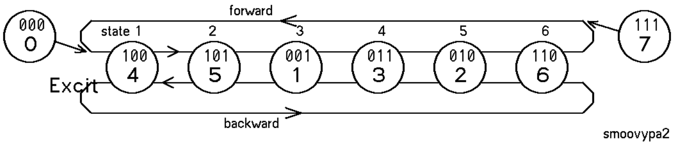
```
Beg:
        Clr         Excit
 Loop:
        Move        Excit,W
        Call        TaForward
        Move        W,Excit
; Superpose other bits to be written on the port
        Move        W,PortB
        Jump        Loop

.macro  d           ; prepare data table
        RetMove     #2´%1,W
.endmacro

Routine Table Motor
TaForward:
        And         #2´111,W
        Add         W,PCL
        d           001            ; any initial valid value
        d           101            ;  001  -->  101   bM2
                                   bM1 bM0
        d           011            ; 010 --> 011
        d           001            ; 011 --> 001
        d           110            ; 100 --> 110
        d           100            ; 101 --> 100
        d           010            ; 110 --> 010
```

4. Bidirectional control

Changing direction may be implemented with two separate tables, with the selection of the table being made according to a variable or a flag. It is simpler, though, to have a single table which includes two 3-bit excitation values corresponding to forward and backward rotation (figure 4).



smoovypa2 *Fig. 4 Bidirectional transition diagram*

The corresponding excitation table can be written explicitly if a macro is defined to get the correct set of bits at the right place. Due to the existence of a Swap instruction, the forward value is placed in the upper 4-bits and the backward in the lower 4-bits.

The program loop is quite simple: according to the direction bit "bBack" in a variable named here "MotorStatus" (could be the same as "Excit", where several bits are free), swapping occurs and the motor can change direction at any state (if the motor speed allows it).

```
        MOVLW   1
        MOVWF   EXCIT
LOOP
        MOVF    EXCIT,W
        CALL    TABIDIR
        MOVWF   EXCIT
        BTFSS   MOTORSTATUS,bBACKWARD
        SWAPF   EXCIT
        MOVF    EXCIT,W
        ANDLW   B'111'
        MOVWF   PORTB
        GOTO    LOOP



TABIDIR
        ANDLW   B'111'
        ADDWF   2
        RETLW   1*16+1
        RETLW   3*16+5
        RETLW   6*16+3
        RETLW   2*16+1
        RETLW   5*16+6
        RETLW   1*16+4
        RETLW   4*16+2
        RETLW   0*16+0
```

Bidirectional control (first solution)

```
        Move          #1,W
        Move          W,Excit         ;  Initialization
 Loop:
        Move          Excit,W
        Call          TaBidir
        Move          W,Excit
        TestSkip,BS    MotorStatus:#bBack
        Swap          Excit
        Move          Excit,W
        And           #2'111,W
; Superpose other bits to be written on the port
        Move          W,PortB
        Jump          Loop

.macro                  dd            ; prepare a table 0xxx0y
        RetMove   #%1*(2**4)+%2,W
.endmacro

TaBidir:
        And           #2'111,W
        Add           W,PCL
        dd            1,1             ; not supposed to get th
        dd            3,5             ; forward 1 ->3 /backw
        dd            6,3
        dd            2,1
        dd            5,6
        dd            1,4
        dd            4,2
        dd            0,0
```

It is in fact faster to have two consecutive tables, and switch according to the "bBack" bit, stored as bit 3 within "Excit".

```
LOOP
        MOVF    EXCIT,W
        CALL    TABIDIR
        MOVWF   EXCIT
        ANDLW   B'111'

        MOVWF   6
; ...
        GOTO    LOOP



TABIDIR
        ANDLW   B'1111'
        ADDWF   2
        RETLW   1
        RETLW   3
        RETLW   6
        RETLW   2
        RETLW   5
        RETLW   1
        RETLW   4
        RETLW   1

        RETLW   1
        RETLW   5
        RETLW   3
        RETLW   1
        RETLW   6
        RETLW   4
        RETLW   2
        RETLW   1
```

Bidirectional control (second solution

```
Loop:
        Move          Excit,W
        Call          TaBidir
        Move          W,Excit
        And           #2'111,W
; Superpose other bits to be written on the port
        Move          W,PortB
; ...
        Jump          Loop

.macro                  dd            ; prepare a table 0xxx0y
        RetMove   #%1,W
.endmacro

TaBidir:
        And           #2'1111,W
        Add           W,PCL
        dd            1               ; not valid, arbitrary ne
        dd            3               ; forward 1 ->3
        dd            6
        dd            2
        dd            5
        dd            1
        dd            4
        dd            1               ; not valid
; When bBack bit is active
        dd            1               ; not valid
        dd            5               ; backward 1 -> 5
        dd            3
        dd            1
        dd            6
        dd            4
        dd            2
        dd            1
```

5. Synchronous programming

Getting step delays from a waiting loop is only possible for simple test programs. Interrupts are not efficient, if supported, with microcontrollers. The solution is to do synchronous programming, where all the operations the program has to do are selected within a loop of constant duration. Motors and devices with precise timings are controlled every loop or every n loops. Other tasks are scheduled according to the previous task, to their priority, or to the raising of flags requesting operation. More details are given in [Nicoud98].

smoopfm6 \colleplan =1il:SMOOPFM6

Fig. 5 Synchronous programming action sequence

Rewriting the previous program for the step control in the 100 μs loop gives:

```
LOOP                                       Loop:                      ; Excecuted every x µs
         DECFSZ    STEPPERIOD                       DecSkip,EQ       StepPeriod
         GOTO      DOSTEP                           Jump             DoStep
         MOVLW     PERIOD                           Move             #Period,W
         MOVWF     STEPPERIOD                       Move             W,StepPeriod
         GOTO      NEXT                             Jump             Next
DOSTEP                                     DoStep:
         MOVF      EXCIT,W                          Move             Excit,W
         CALL      TAFORWARD                        Call             TaForward
         MOVWF     6                                Move             W,PortB
NEXT                                       Next:                      ; continuation
```

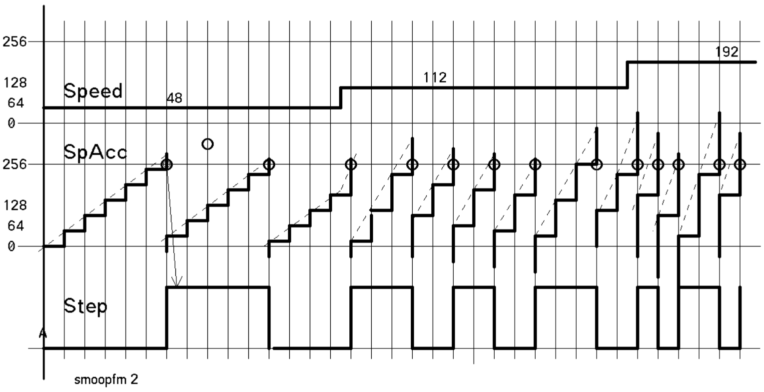
More information on synchronour programming can be found on www.didel.com/DopiSync.pdef.

6. Variable frequency control

In order to modify the rotation speed of the motor, one usually alters the delay between steps, which is the period. Linear period duration modification provides a non-constant acceleration, but the effect is usually insignificant.

A better and frequently simpler solution is to define a speed variable which represents the frequency of the steps. This has the advantage of allowing synchronous programming. At every loop (e.g. every 200 μs), the speed "Speed" is added to a counter "SpAcc". If the counter overflows, a step must be made. With an 8-bit counter, minimum speed (= 1) corresponds to a 51.2 ms step period (200 μs loop), which is about 3 turns per second. Theoretical maximum speed is 255 for a 200 μs period (but there is a 400 μs step every 256 steps), which is about 60,000 RPM.

With both the period approach and the speed approach, the digitalization problem is bad at high speed (compared to processor speed). It is safer to use the available faster PIC or Scenix processors and work with 16-bit precision. Our examples will be given with 8 bits, assuming a speed value between 1 and 50 (20% jitter).



smoopfm2

Fig. 6 Step frequency proportional to "Speed" variable

One may hesitate to clear the SpAcc register when it overflows. The jitter will be reduced, but this jitter generates an average speed with a finer resolution. If SpAcc is reset at every step, several increment values are ignored. For instance, there is no change in speed between 32 and 37, since 32 x 8 = 256 and 37 x 7 = 259.

If the minimum speed value is too fast for some application, it is easy to increase the synchronous loop duration, or to use a 16-bit SpAcc register, with the advantage of a wide speed range with a short loop producing minimal jitter.

```
OP1   MOVF     SPEED,W
      ADDWF    SPACC
      BTFSS    3,0
      GOTO     OP2
      CLRF     SPACC
      MOVF     EXCIT,W
      CALL     TAFORWARD
      MOVWF    EXCIT
      MOVWF    6
OP2   ; continuation
```

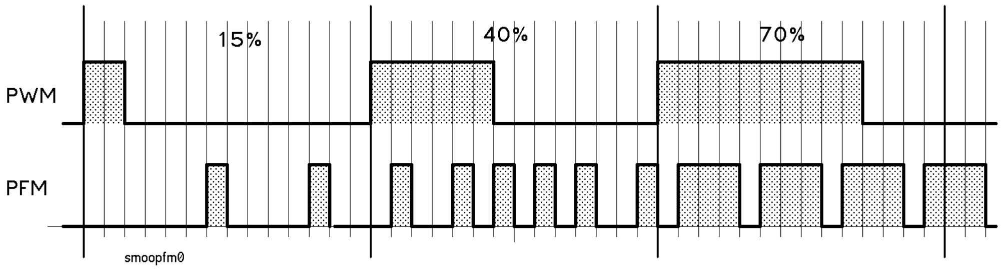
```
; Excecuted every 200 µs (compare with program module on
page xx)
Op1:  Move        Speed,W
      Add         W,SpAcc
      Skip,CS
      Jump        Op2            ; No step if no overflow
        Clr         SpAcc          ; optional
      Move        Excit,W
      Call        TaForward
      Move        W,Excit
      Move        W,PortB
Op2:              ; continuation
```

This program module does not execute in the same amount of time when no step is executed. If required, it is easy to add a jump and several no-op instructions.

7. Variable frequency with PFM transitions

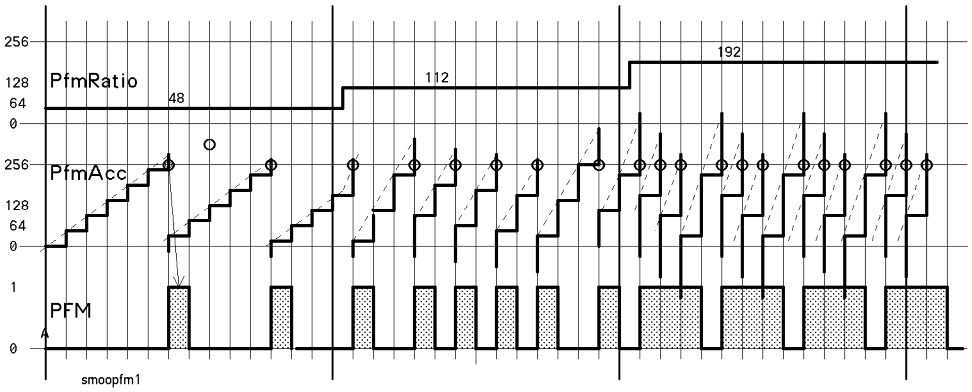
A major difficulty with synchronous motors is that they should be powered with sine waves. At high speed (10000 RPM for an unloaded 3mm smoovy), the motor´s inertia smooths things out, and so square waves are acceptable. O. Matthey has studied the motor´s dynamics and proposes the PFM scheme with trapezoidal ramps as the best solution on the PIC for smoothing rotational movement. Even with PWM hardware, smoothing the steps of a stepping/synchronous motor at smoovy speed would not be easy.

PWM is well known: motor phases receive pulses with a fixed period but a variable width (figure 7a). PFM, on the other hand, uses a fixed positive or negative pulse length and a variable repetition period (figure 7b). When not implemented in hardware (no PIC does this), software PWM implementations require two counters and are tricky to insert in a synchronous programming concept (difficulty also arises when the PWM ratio is 0 or 1).



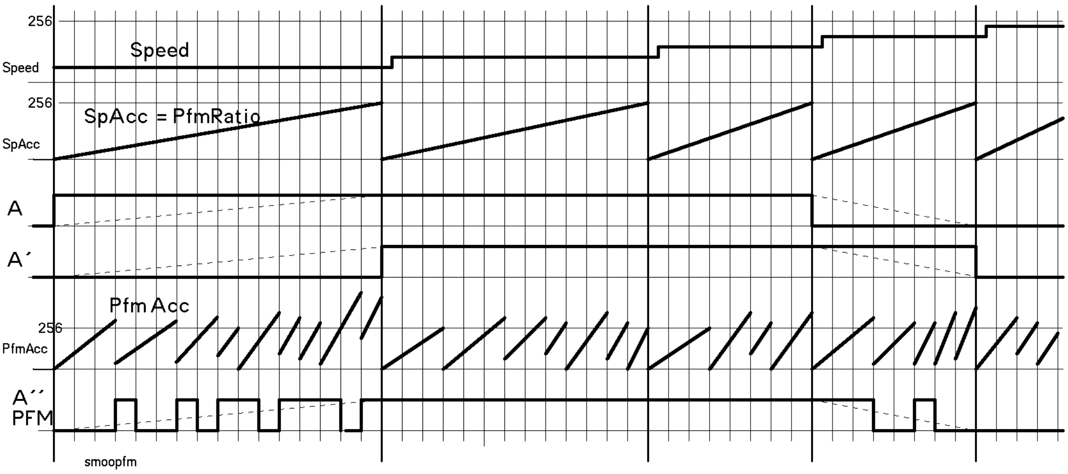
smoopfm0 *Fig. 7 PWM versus PFM*

PFM is easy to implement and is specially suited for synchronous programming or when a fast timer interrupt (e.g. 200 µs) is programmed: the PFM ratio is a value added at every interrupt to a counter PfAcc. When this counter overflows (carry set), the next value is sent to the motor phase. Otherwise (carry clear), the previous value is taken (figure 8).



smoopfm1 *Fig. 8 PFM generation*

The next figure shows approximately how one phase will be switched at every transition, in the case of a very fast speed increase. The SpAcc value is taken as a PfmRatio variable (trapezoidal waveform) and added to the PfAcc counter. When this counter overflows, the next step value is sent to the motor phase.



smoopfm

Fig. 9 PFM interpolation between phase transition

```
          MOVLW     INISPEED
          MOVWF     SPEED
          CLRF      SPACC
          CLRF      PFMACC

LOOP

          MOVF      SPACC,W
          ADDWF     PFMACC
          MOVF      EXCIT,W
          BTFSS     3,0
          SWAPF     EXCIT,W

          ANDLW     B´1111´
          MOVWF     6

          MOVF      SPEED,W
          ADDWF     SPACC
          BTFSS     3,0
          GOTO      NOSTEP

          MOVF      EXCIT,W
          CALL      TAFORWARD
          MOVWF     EXCIT

          GOTO      OP2

NOSTEP
          MOVLW     3
          MOVWF     C1
A         DECFSZ    C1
          GOTO      A

OP2
; ...
          GOTO      LOOP



TAFORWARD
          ANDLW     B´111´
          ADDWF     2
          RETLW     0*16+1
          RETLW     1*16+5
          RETLW     2*16+3
          RETLW     3*16+1
          RETLW     4*16+6
          RETLW     5*16+4
          RETLW     6*16+2
          END
```

```
; End of program initialization
          Move        #IniSpeed,W
          Move        W,Speed
          Clr         SpAcc
          Clr         PfmAcc
; Endless synchronous loop
Loop:                     ; Fixed timing task
           ; Execute a PFM microstep  26 µs duration (16C84/
          Move        SpAcc,W
          Add         W,PfmAcc
          Move        Excit,W
          Skip,CS
          Swap        Excit,W          ; CC, takes previous val
          And         #2´111,W
; Superpose other bits to be written on the port
          Move        W,PortB
; next motor step?
          Move        Speed,W
          Add         W,SpAcc          ; PfmRatio
          Skip,CS
          Jump        NoStep$
             Clr         SpAcc            ; optional
             Clr         PfmAcc           ; optional
          Move        Excit,W
          Call        TaForward
          Move        W,Excit
          Jump        Op2

NoStep$:                                  ; Duration compensatio
          Move        #3,W
          Move        W,C1
    A$:DecSkip,EQ     C1
          Jump        A$

Op2:  ; e.g. control a second motor at different speed
; ...
          Jump        Loop
```

Module **Motor table** **(unidirectional)**

```
.macro                dd               ; prepare a table 0xxx0y
          RetMove     #%1*(2**4)+%2,W
.endmacro
TaForward:
          And         #2´111,W
          Add         W,PCL
          dd          0,1
          dd          1,5              ; "present,next" motor e
          dd          2,3
          dd          3,1
          dd          4,6
          dd          5,4
          dd          6,2
```

J.D. Nicoud, August 1998