MOTOROLA
SEMICONDUCTOR
PROGRAMMING NOTE

Order this document
by TPUPN13/D

STEPPER MOTOR (SM) TPU Function

By Alphonso Gonzalez

1 Functional Overview

The stepper motor control function (SM) accelerates and decelerates a stepper motor linearly. Up to 14 step rates can be used. SM allows a stepper motor to rotate continuously or be used for discrete positioning. The CPU provides the desired step position in a 16-bit parameter, and the TPU steps the motor to the desired position using an acceleration/deceleration profile. The parameter indicating the desired position can be changed by the CPU while the TPU is stepping the motor. The algorithm can change control strategy each time a new step command is received.

A 16-bit parameter initialized by the CPU for each channel defines the output state of the pin. The bit pattern written by the CPU defines the method of stepping, such as full stepping or half stepping. With each transition, the 16-bit parameter rotates one bit. The period of each transition is defined by the programmed step rate.

2 Detailed Description

Any sequential group of up to eight channels can generate the control logic necessary to drive a stepper motor. A group of two or four TPU channels is used. There are two types of stepper motor channels: primary and secondary. The lowest numbered channel is the primary channel, or master, and the higher numbered channels are the secondary channels, or slaves. The secondary channels are serviced by the primary channel except during channel initialization.

To issue a step command, the CPU writes the desired step position to DESIRED_POSITION and generates an HSR on the primary channel. CURRENT_POSITION, containing the current step position, is updated continuously by the TPU as the motor is stepped. If DESIRED_POSITION is greater than STEP_RATE_CNT steps from the CURRENT_POSITION, the TPU accelerates the stepper motor by the programmed acceleration rate, steps at the run rate (maximum programmed rate), and then decelerates the stepper motor by the programmed acceleration rate, stopping at the desired position. If DESIRED_POSITION is within STEP_RATE_CNT or fewer steps from the CURRENT_POSITION, the TPU steps the motor to the position at the start/stop rate (minimum programmed rate).

The SM algorithm allows a change of the control strategy every time a new step command is received, i.e., every time DESIRED_POSITION is changed. The advantages of this scheme become obvious when, for example, DESIRED_POSITION is changed from step 50 to step 10, while stepping has proceeded from step 10 to step 30. Upon the change of DESIRED_POSITION, the TPU decelerates the stepper motor, changes the direction of step, accelerates to the run rate, and decelerates to position 10.

The time period between steps (P) is defined by the following:

$$P(r) = K1 - K2 * r$$

where r is the current step rate (1–14), and K1 and K2 are programmable constants.

STEP_CNTL0 and STEP_CNTL1 define the linear acceleration rate. These parameters relate to K1 and K2 as shown in the following equation.

$$K1 = STEP\_CNTL1 - STEP\_CNTL0$$

$$K2 = STEP\_CNTL0$$

Thus,

$$P(r) = STEP\_CNTL1 - STEP\_CNTL0 * (1 + r)$$

Timer TCR1 is used for matching; therefore, the weighting of the least significant bits (LSB) of STEP_CNTL0 and STEP_CNTL1 is defined to be equal to that of TCR1.

From one to fourteen step rates can be implemented. Two cases exist: STEP_RATE_CNT equals one, or STEP_RATE_CNT is more than one but less than 15. In both cases, when the host provides a new desired position and issues a step request, a delay of P(1) expires before the first step is taken. This delay ensures that the final step of the motor, due to a previous step request, is not disturbed before a time of P(1) has expired. P(1) is also the last programmed step rate used when decelerating to the final step position and the step rate at which a change of direction occurs. When STEP_RATE_CNT is one, P(1) represents the start/stop rate as well as the run rate.

When STEP_RATE_CNT is greater than one, P(1) is the step rate at which a change of direction in stepping will occur, P(2) is the start rate, and $P_{(STEP\_RATE\_CNT)}$ is the run rate. When the condition for changing step direction arises, the stepper motor decelerates to step rate P(1), and the next step is taken in the opposite direction at step rate P(2). How these parameters are used is an important consideration when determining K1 and K2 constants.

PIN_CONTROL associated with each SM channel determines the direction of stepping and whether full or half stepping is to be used. Whenever DESIRED_POSITION is greater than CURRENT_POSITION, the bit pattern of PIN_CONTROL from least significant bit (LSB) to most significant bit (MSB) defines the output levels at the programmed step rates. Whenever DESIRED_POSITION is less than CURRENT_POSITION, the bit pattern of PIN_CONTROL from MSB to LSB defines the output levels at the programmed step rates.

During initialization, SM employs CHANNEL_CONTROL to configure the channel, using the data supplied by the host CPU. The TPU then fills CHANNEL_CONTROL with a copy of PIN_CONTROL for configuring the pin level. With each step taken, another bit of the copy of PIN_CONTROL in CHANNEL_CONTROL is used to configure the pin level, then CHANNEL_CONTROL is rotated in the appropriate direction and written back. After rotating 16 bits to the left or right, CHANNEL_CONTROL is again updated with PIN_CONTROL.

2.1 Stepper Motor Initialization

In the following example, the stepper motor is assumed to be controlled by channels 4–7. Channel 4 is considered the master channel and channels 5, 6, and 7 are the secondary channels. The user should do the following:

1. Set each channel's function select register to the SM algorithm (write the SM function select value to address $FFFE10).
2. Write the pin control bit pattern to each channel's pin control parameter (addresses $YFFF42, $YFFF52, $YFFF62, and $YFFF72). For full stepping, the parameters may be $CCCC, $3333, $9999, and $6666 for channels 4, 5, 6, and 7, respectively.
3. Set each channel control parameter (addresses $YFFF40, $YFFF50, $YFFF60, and $YFFF70) to operate with the appropriate TCR (typically TCR1) and configure pins as outputs. Force each initial pin state to be the same as bit 15 of the respective pin control parameter.
4. In the master channel, set the current position parameter to the presumed midpoint location (address $YFFF44).
5. Set the next step rate to one and the modulo count to zero in the master channel ($0001 to address $YFFF48).

6. Set the last secondary channel number to seven (for channel number 7) and the step count to the number of different step rates (1 to 14) in the master channel ($0x07 to address $YFFF4A).
7. Set the step control parameters in the first secondary channel (channel number 5, addresses $YFFF54, $YFFF56).

   STEP_CNTL0 is the difference between the time interval of two adjacent steps during acceleration or deceleration (assuming the step rate count is > 1). With ten step rates, the period of the longest time interval is 4 ms, followed by intervals of 3.8, 3.6, 3.4, 3.2, 3.0, 2.8, 2.6, 2.4 ms and the rest of the intervals (until deceleration begins) are 2.2 ms between each step. The change in step rate is therefore 0.2 ms, and assuming that TCR1 increments every 10 µs, STEP_CNTL0 is set to (0.2 ms/ 10 µs) = 20 TCR counts.

   STEP_CNTL1 is calculated from equation P(r), as follows:

   $$P(r) = STEP\_CNTL1 - STEP\_CNTL0 * (1 + r)$$

   $$P(r) = STEP\_CNTL1 - 20 * (1 + r)$$

   at r = 1; P(1) = 4.0 ms = 400 TCR counts

   $$400 = STEP\_CNTL1 - (20 * 2)$$

   $$STEP\_CNTL1 = 400 + (20 * 2)$$

   $$STEP\_CNTL1 = 440$$
8. If interrupts are desired, then set the interrupt enable bit for channel 4 by performing an OR between the contents of address $YFFE0A and the value $0010.
9. Execute an HSR for initialization of each channel by writing $AA00 to address $YFFE1A.
10. Turn on each channel by writing to the channel priority register (For mid priority, write $AA00 to address $YFFE1E).

The channels are now configured for operation. The initialization state sets the outputs to their initial levels. A step request is made by writing the new step destination into the desired position parameter of the master channel (address $YFFF46) and by requesting a step from the HSR register ($0300 to address $YFFE1A).

Since stepper motor position is always relative to a start point, the motor is usually positioned or restored to a zero reference location. One technique is to initialize the desired position to zero, and the present position to beyond the maximum position, thereby causing the motor to step against a fixed stop from whatever location it is in when initialization takes place. Another method requires a zero-point switch or sensor. This positional reset is accomplished by interrupts. The master channel for the stepper motor algorithm is set to generate interrupts, and software flags keep track of the current operation. The algorithm generates an interrupt after initialization and after every step request.

2.2 Stepper Motor Interrupts

Following is a sequence of pseudocode that uses a separate zero-point sensor to cause interrupts, saves the position of the stepper corresponding to the zero point, and sets the flag indicating that the motor has been zeroed (and clears the interrupt enable for the master stepper channel). Normal operation does not cause interrupts, but an attempt to return to the zero point causes an interrupt.

```
Clear interrupt flag
IF the stepper channel has been initialized THEN continue
ELSE
        Set initialized flag
        IF at point zero
               THEN Set zeroed flag
                           Save zero location
```

```
                         Quit
        ELSE Step request down
                     Set seek flag down
                     Quit
        END IF
END IF
IF the stepper motor has been zeroed THEN continue
ELSE                     (Note 1)
        IF the seek flag is up
               THEN Step request down
                         Set seek flag down
                         Quit
        ELSE Step request up
                     Set seek flag up
                     Quit
        END IF
END IF
IF the stepper motor is still not zeroed
THEN                     (Note 2)
               Clear zeroed flag
        IF the seek flag is up
        THEN Step request down
                     Set seek flag down
                     Quit
        ELSE Step request up
                     Set seek flag up
                     Quit
        END IF
ELSE                 **No reason for interrupt**
        Quit
END IF
```

NOTES:

1. Initialized but not zeroed. The interrupt must indicate that the first step request down is complete without encountering the zero switch (or possibly has already sought both directions and not encountered the zero sensor). The motor may be too far below the switch, and should, therefore, seek upwards (or down again).
2. This is an error case. Either the interrupt has no reason, or the zero point is not yet found (the motor is lost). In that case, begin seeking for the zero point.

3 Function Code Size

Total TPU function code size determines what combination of functions can fit into a given ROM or emulation memory microcode space. SM function code size is:

63 μ instructions + 6 entries = **69 long words**

4 Function Parameters

This section provides detailed descriptions of discrete input/output function parameters stored in channel parameter RAM. **Figure 1** shows TPU parameter RAM address mapping. **Figure 2** shows the parameter RAM assignment used by the SM function. In the diagrams, Y = M111, where M is the value of the module mapping bit (MM) in the system integration module configuration register (Y = $7 or $F).

Channel Number	Base Address	Parameter Address							
		0	1	2	3	4	5	6	7
0	$YFFF##	00	02	04	06	08	0A	—	—
1	$YFFF##	10	12	14	16	18	1A	—	—
2	$YFFF##	20	22	24	26	28	2A	—	—
3	$YFFF##	30	32	34	36	38	3A	—	—
4	$YFFF##	40	42	44	46	48	4A	—	—
5	$YFFF##	50	52	54	56	58	5A	—	—
6	$YFFF##	60	62	64	66	68	6A	—	—
7	$YFFF##	70	72	74	76	78	7A	—	—
8	$YFFF##	80	82	84	86	88	8A	—	—
9	$YFFF##	90	92	94	96	98	9A	—	—
10	$YFFF##	A0	A2	A4	A6	A8	AA	—	—
11	$YFFF##	B0	B2	B4	B6	B8	BA	—	—
12	$YFFF##	C0	C2	C4	C6	C8	CA	—	—
13	$YFFF##	D0	D2	D4	D6	D8	DA	—	—
14	$YFFF##	E0	E2	E4	E6	E8	EA	EC	EE
15	$YFFF##	F0	F2	F4	F6	F8	FA	FC	FE

— = Not Implemented (reads as $00)

Figure 1 TPU Channel Parameter RAM CPU Address Map

Figure 2 shows all of the host interface areas for the SM function, described in **5 Host Interface to Function**, as well as the parameters, addresses, reference times, and reference sources. This segment lists and defines the parameters for all modes of the SM time function.

PRIMARY CHANNEL

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$YFFFW0								CHANNEL_CONTROL								
$YFFFW2	PIN_CONTROL															
$YFFFW4	CURRENT_POSITION															
$YFFFW6	DESIRED_POSITION															
$YFFFW8	0	0	0	0	MOD_CNT				0	0	0	0	NEXT_STEP_RATE			
$YFFFWA	0	0	0	0	STEP_RATE_CNT				0	0	0	0	LAST_SEC_CHAN			

SECONDARY CHANNEL 1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$YFFFW0								CHANNEL_CONTROL								
$YFFFW2	PIN_CONTROL															
$YFFFW4	STEP_CNTL0															
$YFFFW6	STEP_CNTL1															
$YFFFW8																
$YFFFWA																

SECONDARY CHANNELS 2 — 7

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$YFFFW0								CHANNEL_CONTROL								
$YFFFW2	PIN_CONTROL															
$YFFFW4																
$YFFFW6																
$YFFFW8																
$YFFFWA																

Y= Channel number

Parameter Write Access:

- Written by CPU
- Written by TPU
- Written by CPU and TPU
- Unused parameters

Figure 2 Function Parameter RAM Assignment

4.1 CHANNEL_CONTROL

The CPU must write CHANNEL_CONTROL for the primary and for all secondary channels prior to initialization in order to establish initial PSC, PAC, and TBS fields. The PSC field forces the output level of the pin directly without affecting the pin action control latches and should be configured to force the same level as bit 15 of PIN_CONTROL. The PAC field is not used. The following table defines the allowable data for this parameter. CHANNEL_CONTROL PSC field should be initialized to the same level as the MSB of PIN_CONTROL.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT USED							TBS				PAC			PSC	

SM CHANNEL_CONTROL Options

TBS	PAC	PSC	Action	
8 7 6 5	**4 3 2**	**1 0**	**Input**	**Output**
		0 1 1 0 1 1	— — —	Force Pin High Force Pin Low Do Not Force Any State
0 1 x x 0 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1 1 x x x			 — — — — Do Not Change TBS	**Output Channel** Capture TCR1, Match TCR1 Capture TCR1, Match TCR2 Capture TCR2, Match TCR1 Capture TCR2, Match TCR2 Do Not Change TBS

After configuring the channel latches in initialization, the TPU fills CHANNEL_CONTROL with a copy of PIN_CONTROL, which later is used to configure the pin level. Each time the TPU services an SM channel, it rotates the content of CHANNEL_CONTROL in the appropriate direction and then uses bit 15 of CHANNEL_CONTROL to configure the pin level. After CHANNEL_CONTROL is rotated 16 bits to the left or right, SM again copies PIN_CONTROL into CHANNEL_CONTROL.

4.2 PIN_CONTROL

PIN_CONTROL contains data representing the sequence of pin transitions. This 16-bit parameter must be initialized by the CPU for all channels used for stepper motor control. The bit pattern of the parameter indicates the sequence of output states of the pin. As an example for full-stepping sequence, typical parameters for two channels are $3333 and $9999. For half-stepping sequence, typical parameters are $E0E0, $0E0E, $8383, and $3838 for four successive channels, respectively. An illustrated example of PIN_CONTROL data follows for full stepping a two-phase motor.

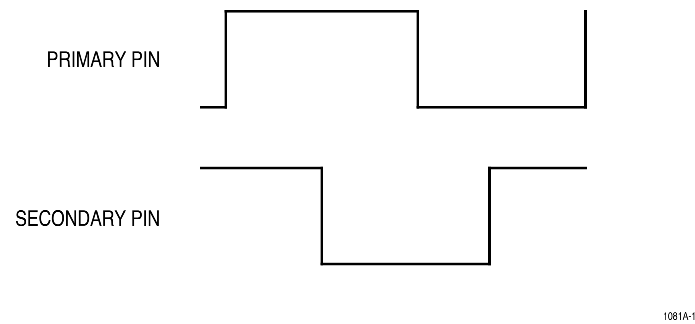
4.2.1 Primary Channel

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

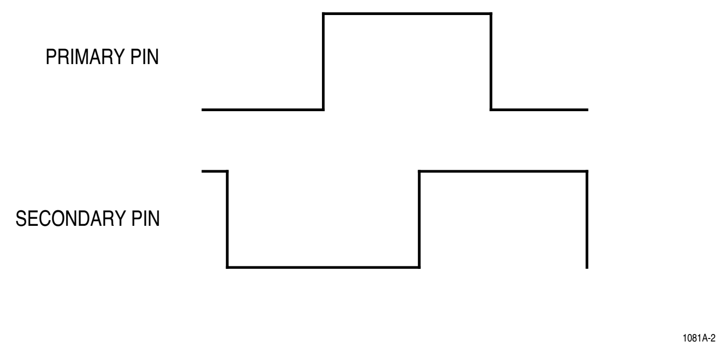
4.2.2 Secondary Channel

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1

When DESIRED_POSITION is greater than CURRENT_POSITION, the bit pattern from LSB to MSB defines the output levels at the programmed step rates, i.e., bit 0 is moved into bit 15, to define the output. The output pins are sequenced as shown.



When DESIRED_POSITION is less than CURRENT_POSITION, the bit pattern from MSB to LSB defines the output levels at the programmed step rates, i.e., bit 14 is moved into bit 15 to define the output. The output pins are sequenced as shown.



4.3 CURRENT_POSITION

CURRENT_POSITION contains the current step position of the stepper motor. This unsigned parameter is incremented or decremented each time CHANNEL_CONTROL is rotated. In other words, in a full-step configuration, each count represents a full step; in a half-step configuration, each count represents a half step.

4.4 DESIRED_POSITION

DESIRED_POSITION contains the desired position of the stepper motor and is updated by the CPU. The range for DESIRED_POSITION is $0 to ($FFFF–STEP_RATE_CNT), unsigned.

4.5 LAST_SEC_CHAN

LAST_SEC_CHAN contains the channel number of the last secondary channel among the channels to be sequentially grouped, beginning with the primary channel, to synthesize the SM algorithm. The CPU updates this parameter. The upper four bits of this parameter must be set to zero.

4.6 STEP_RATE_CNT

STEP_RATE_CNT contains the number of step rates used in the acceleration/deceleration profile. The CPU updates this parameter to a value between 1 and 14 ($1–$E).

4.7 NEXT_STEP_RATE

NEXT_STEP_RATE contains the step rate to be programmed after the current step is complete. This parameter must be initialized to one by the CPU and is updated subsequently by the TPU.

4.8 MOD_CNT

MOD_CNT indicates the four-bit modulo count for the number of rotations that have occurred. MOD_CNT must be initialized to $0 by the CPU and subsequently is updated by the TPU. This parameter is incremented and decremented concurrently with CURRENT_POSITION. When MOD_CNT equals zero, PIN_CONTROL is copied to CHANNEL_CONTROL.

4.9 STEP_CNTL0 and STEP_CNTL1

STEP_CNTL0 and STEP_CNTL1, located in the first secondary channel, are updated by the CPU, and specify the acceleration/deceleration rate described earlier.

All 16 bits of STEP_CNTL0 and STEP_CNTL1 may be used in defining constants K1 and K2; however, the time period between steps P(r) must be less than $8000 for all programmed step rates.

5 Host Interface to Function

This section provides information concerning the TPU host interface to the function. **Figure 3** is a TPU address map. Detailed TPU register diagrams follow the figure. In the diagrams, Y = M111, where M is the value of the module mapping bit (MM) in the system integration module configuration register (Y = $7 or $F).

Address	15 8	7 0
$YFFE00	TPU MODULE CONFIGURATION REGISTER (TPUMCR)	
$YFFE02	TEST CONFIGURATION REGISTER (TCR)	
$YFFE04	DEVELOPMENT SUPPORT CONTROL REGISTER (DSCR)	
$YFFE06	DEVELOPMENT SUPPORT STATUS REGISTER (DSSR)	
$YFFE08	TPU INTERRUPT CONFIGURATION REGISTER (TICR)	
$YFFE0A	CHANNEL INTERRUPT ENABLE REGISTER (CIER)	
$YFFE0C	CHANNEL FUNCTION SELECTION REGISTER 0 (CFSR0)	
$YFFE0E	CHANNEL FUNCTION SELECTION REGISTER 1 (CFSR1)	
$YFFE10	CHANNEL FUNCTION SELECTION REGISTER 2 (CFSR2)	
$YFFE12	CHANNEL FUNCTION SELECTION REGISTER 3 (CFSR3)	
$YFFE14	HOST SEQUENCE REGISTER 0 (HSQR0)	
$YFFE16	HOST SEQUENCE REGISTER 1 (HSQR1)	
$YFFE18	HOST SERVICE REQUEST REGISTER 0 (HSRR0)	
$YFFE1A	HOST SERVICE REQUEST REGISTER 1 (HSRR1)	
$YFFE1C	CHANNEL PRIORITY REGISTER 0 (CPR0)	
$YFFE1E	CHANNEL PRIORITY REGISTER 1 (CPR1)	
$YFFE20	CHANNEL INTERRUPT STATUS REGISTER (CISR)	
$YFFE22	LINK REGISTER (LR)	
$YFFE24	SERVICE GRANT LATCH REGISTER (SGLR)	
$YFFE26	DECODED CHANNEL NUMBER REGISTER (DCNR)	

Figure 3 TPU Address Map

CIER — Channel Interrupt Enable Register **$YFFE0A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15	CH 14	CH 13	CH 12	CH 11	CH 10	CH 9	CH 8	CH 7	CH 6	CH 5	CH 4	CH 3	CH 2	CH 1	CH 0

CH	Interrupt Enable
0	Channel interrupts disabled
1	Channel interrupts enabled

CFSR[0:3] — Channel Function Select Registers **$YFFE0C – $YFFE12**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CFS (CH 15, 11, 7, 3)				CFS (CH 14, 10, 6, 2)				CFS (CH 13, 9, 5, 1)				CFS (CH 12, 8, 4, 0)			

CFS[4:0] — SM Function Number (Assigned during microcode assembly)

HSQR[0:1] — Host Sequence Registers **$YFFE14 – $YFFE16**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15, 7		CH 14, 6		CH 13, 5		CH 12, 4		CH 11, 3		CH 10, 2		CH 9, 1		CH 8, 0	

No host sequence encodings are implemented for the SM function.

HSRR[1:0] — Host Service Request Registers **$YFFE18 – $YFFE1A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15, 7		CH 14, 6		CH 13, 5		CH 12, 4		CH 11, 3		CH 10, 2		CH 9, 1		CH 8, 0	

CH	Initialization
00	No Host Service Request
01	No Host Service Request
10	Initialization
11	Step Request

CPR[1:0] — Channel Priority Registers **$YFFE1C – $YFFE1E**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15, 7		CH 14, 6		CH 13, 5		CH 12, 4		CH 11, 3		CH 10, 2		CH 9, 1		CH 8, 0	

CH	Channel Priority
00	Disabled
01	Low
10	Middle
11	High

CISR — Channel Interrupt Status Register **$YFFE20**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15	CH 14	CH 13	CH 12	CH 11	CH 10	CH 9	CH 8	CH 7	CH 6	CH 5	CH 4	CH 3	CH 2	CH 1	CH 0

CH	Interrupt Status
0	Channel interrupt not asserted
1	Channel interrupt asserted

6 Function Configuration

The CPU configures both the primary and secondary channels as follows:

1. Updates channel parameters for all primary and secondary channels;
2. Generates an HSR %10 on each channel for initialization; and
3. Enables channel servicing by assigning a high, middle, or low priority.

The TPU then initializes the primary and each secondary channel. The CPU should monitor the HSR register until the TPU clears the service request of all SM channels to 00 before changing any parameters or before issuing a new service request to the primary channel. Next, the CPU issues a request for stepping by updating DESIRED_POSITION with the desired step position and by generating an HSR %11 on the primary channel. Further CPU interface with the secondary channels is not required.

7 Performance of Function

Like all TPU functions, SM function performance in an application is to some extent dependent upon the service time (latency) of other active TPU channels. This is due to the operational nature of the scheduler. When more TPU channels are active, performance decreases. However, worst-case latency in any TPU application can be closely estimated. To analyze the performance of an application that appears to approach the limits of the TPU, use the guidelines given in the TPU reference manual and in the table below.

SM State Timing

State Name	Clock Cycles	RAM Accesses	Conditions
S1 *Init*	8	2	—
S2 *Step_Request*	134[1]	15	—
S3 *Right_Rot_Chn_Cntl*	160[1]	21	Motor has decelerated and step direction changed.
	122[1]	14	All other cases
S4 *Left_Rot_Chn_Cntl*	158[1]	21	Motor has decelerated and step direction changed.
	120[1]	14	All other cases

NOTES

1.Includes one master plus one slave:
(a) add 32 clocks and two RAM accesses for each additional slave, and
(b) add (STEP_RATE_CNT ∗ two clocks).

8 Function Example

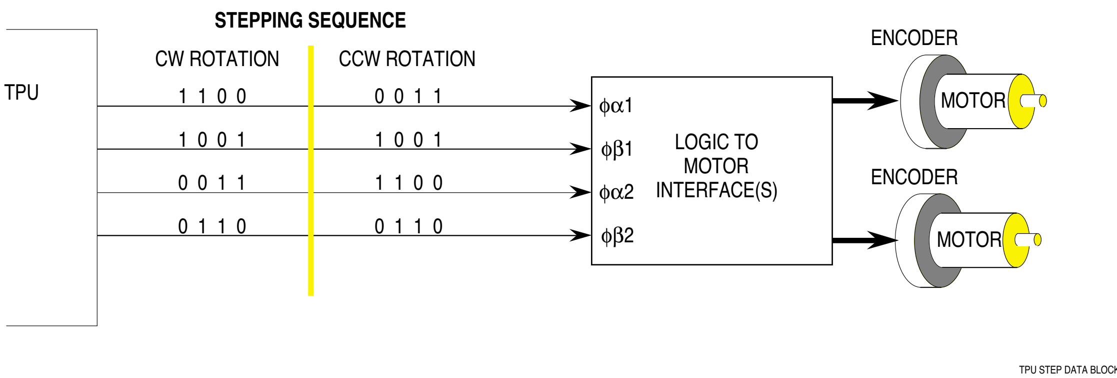
8.1 Continuous Rotation

8.1.1 Introduction

This example illustrates how to continuously rotate a stepper motor using the SM algorithm. It begins by detailing the connections used to interface the motor and concludes with a discussion of programming for continuous rotation. Code listings for modular microcontroller CPU32 and CPU16 modules are included at the end of the example.

8.1.1.1 Logic-to-Motor Interface

The stepper motor used in this example operates on a twelve volt direct current input. The motor can be configured for bipolar operation. This means that voltage of the stepping sequence varies from zero to twelve volts. For a unipolar motor, the voltage varies from a negative voltage to a positive voltage. The stepping sequence followed by the motor is shown below. The two phases of the stepper motor windings correspond to $\phi\alpha$ and $\phi\beta$.



The logic output to the stepper motor varies from zero at the low state (0) to five at the high state (1). The outputs are driven from channels four through seven (TPUCH4 – TPUCH7) of the TPU. Because of the voltage difference between the logic outputs and the voltage requirement of the stepper motor, an interface must be used.

The logic-to-motor interface (LMI), Motorola product DEVB-103, is an evaluation board used to interface logic inputs to motor drives. The LMI contains a complementary H-Bridge. It can be used to interface stepper motors as well as brush DC motors. For further information on the device, refer to Motorola Application Note AN1300.

The motor is connected as shown below. The interface requires two LMI devices. Each LMI drives a different phase of the stepper motor.

8.1.1.2 Programming for Continuous Rotation

Programming the TPU for stepper motor control requires two procedures: initialization and step requests. These procedures are accomplished by moving information into the appropriate parameter RAM locations and then issuing a host service request for initialization or step request.

The channels used for stepping must be adjacent. The primary channel used for stepping should be the lowest. The remaining channels used for stepping are considered secondary channels. In this example, channel four is the primary channel and channels five through seven are the secondary channels.

8.1.1.3 Initialization

The initialization procedure configures the required channels. The parameters STEP_CNTL0 and STEP_CNTL1 are configured for the particular motor used. The DESIRED_POSITION and CURRENT_POSITION are set to be equal. Then, a request for initialization is issued on the SM channels.

The step request portion of the program is accomplished in the CYCLE subroutine. This short subroutine changes the desired position to zero and then issues a step request by moving 11 into the HSR address ($FFFE1A) of the primary channel. The TPU then determines which direction to rotate the stepper motor. Finally, the branch always command causes the program to loop back to the CYCLE subroutine. The effect of looping the program and resetting the DESIRED_POSITION counter is like walking on a treadmill. The TPU attempts to increment the CURRENT_POSITION, but the counter is reset before reaching the DESIRED_POSITION. The result is continuous rotation of the stepper motor.

As long as the difference between the CURRENT_POSITION and DESIRED_POSITION is large enough not to equalize during the loop, the motor will not stop rotating. Also, it is not sufficient to simply change the CURRENT_POSITION or DESIRED_POSITION locations. An HSR for step request must be made so that SM can perform the appropriate action.

8.2 Listing for CPU32-Based Microcontrollers

Objective: This program continually rotates a stepper motor.
Method: The stepper motor is driven by TPUCH4 through TPUCH7 in a full-step sequence using the SM Algorithm.

This program was assembled using the IASM32 assembler available from P&E Microcomputer Systems with the M68332 In-Circuit Debugger. It was run on an M68332EVS and BCC.

```
CFSR2      equ    $fffe10
CFSR3      equ    $fffe12
HSQR0      equ    $fffe14
HSQR1      equ    $fffe16
HSRR0      equ    $fffe18
HSRR1      equ    $fffe1a
CPR1       equ    $fffe1e
PRAM4_0    equ    $ffff40
PRAM4_1    equ    $ffff42
PRAM4_2    equ    $ffff44
PRAM4_3    equ    $ffff46
PRAM4_4    equ    $ffff48
PRAM4_5    equ    $ffff4a
PRAM5_0    equ    $ffff50
PRAM5_1    equ    $ffff52
PRAM5_2    equ    $ffff54
PRAM5_3    equ    $ffff56
PRAM5_4    equ    $ffff58
PRAM5_5    equ    $ffff5a
PRAM6_0    equ    $ffff60
PRAM6_1    equ    $ffff62
PRAM6_2    equ    $ffff64
PRAM6_3    equ    $ffff66
PRAM6_4    equ    $ffff68
PRAM6_5    equ    $ffff6a
PRAM7_0    equ    $ffff70
PRAM7_1    equ    $ffff72
PRAM7_2    equ    $ffff74
PRAM7_3    equ    $ffff76
```

```
PRAM7_4     equu    $ffff78
PRAM7_5     equ     $ffff7a
    ORG     $6000
```

The following line initializes channels 4 through 7 for SM operation

```
MOVE.W #$DDDD,(CFSR2).l   ;CHANNEL FUNCTION SELECT, VALUE MAY DIFFER FOR OTHER MASK SETS
```

The STEPINIT subroutine configures the channels for SM operation. It sets the acceleration constants and initializes the position registers.

```
STEPINIT       MOVE.W #$CCCC,(PRAM4_1).l     ;SET PIN CONTROL 4
               MOVE.W #$3333,(PRAM5_1).l     ;SET PIN CONTROL 5
               MOVE.W #$9999,(PRAM6_1).l     ;SET PIN CONTROL 6
               MOVE.W #$6666(PRAM7_1).l      ;SET PIN CONTROL 7
               MOVE.W #$81,(PRAM4_0).l       ;CHANNEL CONTROL 4
               MOVE.W #$82,(PRAM5_0).l       ;CHANNEL CONTROL 5
               MOVE.W #$81,(PRAM6_0).l       ;CHANNEL CONTROL 6
               MOVE.W #$82,(PRAM7_0).l       ;CHANNEL CONTROL 7
               MOVE.W #$7FFF,(PRAM4_2).l     ;PRESUMED MIDPOINT LOCATION
               MOVE.W #$7FFF,(PRAM4_3).l     ;DESIRED POSITION
               MOVE.W #$1,(PRAM4_4).l        ;NEXT STEP RATE AND MOD CNT
               MOVE.W #$E07,(PRAM4_5).l      ;STEP RATE CNT AND LAST SEC CHN
               MOVE.W #$4D,(PRAM5_2).l       ;STEP CNTL 0
               MOVE.W #$D9F,(PRAM5_3).l      ;STEP CNTL 1
               MOVE.W #$AA00,(HSRR1).l       ;HSR FOR INIT
               MOVE.W #$FF00,(CPR1).l        ;SET CHNS TO HIGH PRIORITY
```

The CYCLE subroutine resets the CURRENT_POSITION address to zero and issues a step request. In effect, DESIRED_POSITION never equals CURRENT_POSITION.

```
CYCLE   MOVE.W #$0,(PRAM4_2).l    ;CURRENT POSITION
        MOVE.W #$300,(HSRR1).l    ;HSR STEP REQUEST
        BRA    CYCLE
```

8.3 LISTING FOR CPU16-BASED MICROCONTROLLERS

Objective: This program continually rotates a stepper motor.

Method: The stepper motor is driven by TPUCH4 through TPUCH7 in a full-step sequence using the SM Algorithm.

This program was assembled on the IASM16 Assembler available with the ICD16 In-Circuit Debugger from P&E Microcomputer Systems. It was run on an MC68HC16Y1EVB.

```
CFSR2          equ    $fe10
CFSR3          equ    $fe12
HSQR0          equ    $fe14
HSQR1          equ    $fe16
HSRR0          equ    $fe18
HSRR1          equ    $fe1a
CPR1           equ    $fe1e
PRAM4_0        equ    $ff40
PRAM4_1        equ    $ff42
PRAM4_2        equ    $ff44
PRAM4_3        equ    $ff46
PRAM4_4        equ    $ff48
PRAM4_5        equ    $ff4a
```

```
PRAM5_0         equ     $ff50
PRAM5_1         equ     $ff52
PRAM5_2         equ     $ff54
PRAM5_3         equ     $ff56
PRAM5_4         equ     $ff58
PRAM5_5         equ     $ff5a
PRAM6_0         equ     $ff60
PRAM6_1         equ     $ff62
PRAM6_2         equ     $ff64
PRAM6_3         equ     $ff66
PRAM6_4         equ     $ff68
PRAM6_5         equ     $ff6a
PRAM7_0         equ     $ff70
PRAM7_1         equ     $ff72
PRAM7_2         equ     $ff74
PRAM7_3         equ     $ff76
PRAM7_4         equ     $ff78
PRAM7_5         equ     $ff7a
```

The following code is included to set up the reset vector ($00000 – $00006). It may be changed for different systems.

```
        ORG     $0000               ;PUT THE FOLLOWING RESET VECTOR INFORMATION
                                    ;AT ADDRESS $00000 OF THE MEMORY MAP
        DW      $0000               ;ZK=0, SK=0, PK=0
        DW      $0200               ;PC=200 -- INITIAL PROGRAM COUNTER
        DW      $3000               ;SP=3000 -- INITIAL STACK POINTER
        DW      $0000               ;IZ=0 -- DIRECT PAGE POINTER
        ORG     $0400               ;BEGIN PROGRAM AT MEMORY LOCATION $0400
```

The following code initializes and configures the system, including software watchdog and system clock.

```
INITSYS                             ;give initial values for extension registers
                                    ;AND INITIALIZE SYSTEM CLOCK AND COP
                LDAB    #$0F
                TBEK                ;POINT EK TO BANK F FOR REGISTER ACCESS
                LDAB    #$00
                TBXK                ;POINT XK TO BANK 0
                TBYK                ;POINT YK TO BANK 0
                TBZK                ;POINT ZK TO BANK 0
                TBSK
                LDD     #$0003      ;AT RESET, THE CSBOOT BLOCK SIZE IS 512K.
                STD     CSBARBT     ;THIS LINE SETS THE BLOCK SIZE TO 64K SINCE
                                    ;THAT IS WHAT PHYSICALLY COMES WITH THE EVB16
                LDAA     #$7F       ;W=0, X=1, Y=111111
                STAA    SYNCR       ;SET SYSTEM CLOCK TO 16.78 MHZ
                CLR     SYPCR       ;TURN COP (SOFTWARE WATCHDOG) OFF,
                                    ;SINCE COP IS ON AFTER RESET
                LDS     #$f000
                LDAB    #$0F
                TBEK                ;USE BANK $0F FOR PARAMETER RAM
```

The following line initializes channels 4 through 7 for SM operation.

```
                LDD #$DDDD
                STD CFSR2           ;CHANNEL FUNCTION SELECT
```

The STEPINIT subroutine configures the channels for SM operation. It sets the acceleration constants and initializes the position registers.

```
STEPINIT        LDD #$CCCC
                STD PRAM4_1       ;SET PIN CONTROL 4
                LDD #$3333
                STD PRAM5_1       ;SET PIN CONTROL 5
                LDD #$9999
                STD PRAM6_1       ;SET PIN CONTROL 6
                LDD #$6666
                STD PRAM7_1       ;SET PIN CONTROL 7
                LDD #$81
                STD PRAM4_0       ;CHANNEL CONTROL 4
                LDD #$82
                STD PRAM5_0       ;CHANNEL CONTROL 5
                LDD #$81
                STD PRAM6_0       ;CHANNEL CONTROL 6
                LDD #$82
                STD PRAM7_0       ;CHANNEL CONTROL 7
                LDD #$7FFF
                STD PRAM4_2       ;PRESUMED MIDPOINT LOCATION
                LDD #$7FFF
                STD PRAM4_3       ;DESIRED POSITION
                LDD #$1
                STD PRAM4_4       ;NEXT STEP RATE AND MOD CNT
                LDD #$E07
                STD PRAM4_5).l;STEP RATE CNT AND LAST SEC CHN
                LDD #$4D
                STD PRAM5_2       ;STEP CNTL 0
                LDD #$D9F
                STD PRAM5_3       ;STEP CNTL 1
                LDD #$AA00
                STD HSRR1         ;HSR FOR INIT
                LDD #$FF00
                STD CPR1          ;SET CHNS TO HIGH PRIORITY
```

The CYCLE subroutine resets the CURRENT_POSITION address to zero and issues a step request. In effect, DESIRED_POSITION never equals CURRENT_POSITION.

```
CYCLE           LDD #$0
                STD PRAM4_2       ;CURRENT POSITION
                LDD #$300
                STD HSRR1         ;HSR STEP REQUEST
                BRA    CYCLE
```

9 SM Algorithm

The following description is provided as a guide only, to aid understanding of the function. The exact sequence of operations in microcode may be different from that shown, in order to optimize speed and code size. TPU microcode source listings for all functions in the TPU function library can be downloaded from the Motorola Freeware bulletin board. Refer to *Using the TPU Function Library and TPU Emulation Mode* (TPUPN00/D) for detailed instructions on downloading and compiling microcode.

The SM time function consists of four states, described below. For clarity, reference is made to internal flags in the following descriptions. These internal TPU control bits are not available to the user. Flag0 determines the direction of rotation of CHANNEL_CONTROL. If negated, a left rotate of

CHANNEL_CONTROL determines the pattern for the output pin level. If asserted, a right rotate of CHANNEL_CONTROL determines the pattern for the output pin level. Flag1 indicates that stepping is in progress.

9.1 State 1: *Init*

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 10xxxx
Match Enable: Disable

Summary:

This state is entered as a result of an HSR %10. In this state, the channel latches are configured using CHANNEL_CONTROL. CHANNEL_CONTROL is subsequently updated by the TPU with PIN_CONTROL.

Algorithm:

Configure the channel latches via CHANNEL_CONTROL
Update CHANNEL_CONTROL with PIN_CONTROL
Clear flag1 and all service requests
Assert interrupt request

9.1.1 State 2: *Step_Request*

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 11xxxx
Match Enable: Disable

Summary:

This state is entered as a result of an HSR %11. During this state, the TPU determines, based on CURRENT_POSITION and DESIRED_POSITION, the direction that the motor should be stepped. Whenever DESIRED_POSITION is less than CURRENT_POSITION, the bit pattern from MSB to LSB defines the output levels at the programmed step rates, i.e., bit 14 is moved into bit 15 to define the output, and flag0 is negated, directing activity to state 3. Whenever DESIRED_POSITION is greater than CURRENT_POSITION, the bit pattern from LSB to MSB defines the output levels at the programmed step rates, i.e., bit 0 is moved into bit 15 to define the output, and flag0 is asserted, directing activity to state 4.

Algorithm:

```
    If (flag1 = 1) then                                        /* is stepping in progress? */
        End                                           /* yes, no need to initiate a step request */
    }
    Else, assert flag1                                         /* no, set stepping flag and begin */
    Save TCR1 time into ERT                         /* TCR1 value to be used in STEP_SETUP */
TEST:
    If (DESIRED_POSITION = CURRENT_POSITION) then
        Clear flag1 and all service requests                                  /* stepping done */
        Assert interrupt request
        End
    }
    Else, if (DESIRED_POSITION > CURRENT_POSITION) then
        Assert flag0                                           /* assert step direction flag */
        Goto RIGHT_ROT
    }
    Else, negate flag0                                         /* negate step direction flag */
    Goto LEFT_ROT
```

9.1.2 State 3: ***Left_Rot_Chn_Cntl***

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 001xx0
Match Enable: Enable

Summary:

This state is entered after a match event while flag0 is negated. In this state, while CURRENT_POSITION is less than DESIRED_POSITION, stepping proceeds in the direction specified by the CHANNEL_CONTROL bit pattern from MSB to LSB.

Algorithm:

```
LEFT_ROT:
    Decrement MOD_CNT
    NEXT_STEP_RATE_TEMP = NEXT_STEP_RATE
    If (CURRENT_POSITION ≥ DESIRED_POSITION + STEP_RATE_CNT) then
        Decrement CURRENT_POSITION
        If (NEXT_STEP_RATE < STEP_RATE_CNT) then
            Increment NEXT_STEP_RATE
        }
        Goto STEP_SETUP                                   /* setup next step */
    }
    Else, if (NEXT_STEP_RATE = 1) then          /*is NEXT_STEP_RATE minimum?*/
        If (CURRENT_POSITION > DESIRED_POSITION) then
            Decrement CURRENT_POSITION
            Goto STEP_SETUP                               /* setup next step */
        }
        Else {
            Goto TEST
        }
    }
    Else, {
        Decrement CURRENT_POSITION
        Decrement NEXT_STEP_RATE
        Goto STEP_SETUP                                   /* setup next step */
    }
```

9.1.3 State 4: ***Right_Rot_Chn_Cntl***

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 001xx1
Match Enable: Enable

Summary:

This state is entered after a match event while flag0 is asserted. In this state, while CURRENT_POSITION is greater than DESIRED_POSITION, stepping proceeds in the direction specified by the CHANNEL_CONTROL bit pattern from LSB to MSB.

Algorithm:

```
RIGHT_ROT
    Increment MOD_CNT
    NEXT_STEP_RATE_TEMP = NEXT_STEP_RATE
    If (DESIRED_POSITION ≥ CURRENT_POSITION + STEP_RATE_CNT) then
        Increment CURRENT_POSITION
        If (NEXT_STEP_RATE < STEP_RATE_CNT) then
            Increment NEXT_STEP_RATE
        }
        Goto STEP_SETUP                                   /* setup next step */
    }
```

```
Else, if (NEXT_STEP_RATE = 1) then                    /*is NEXT_STEP_RATE minimum?*/
    If (DESIRED_POSITION > CURRENT_POSITION) then
        Increment CURRENT_POSITION
        Goto STEP_SETUP                                /* setup next step */
    }
    Else {
        Goto TEST
    }
}
Else {
    Increment CURRENT_POSITION
    Decrement NEXT_STEP_RATE
    Goto STEP_SETUP                                    /* setup next step */
}
```

9.1.4 Step Setup Routine (Subroutine)

```
STEP_SETUP                      /*calculate match time for step; MATCH_TEMP
                                  is a temporary register */
    MATCH_TEMP = ERT + STEP_CNTL1 –
            STEP_CNTL0 ∗ (NEXT_STEP_RATE_TEMP + 1)]
    SHIFT_DIRECTION_TEMP = flag0    /* save flag0 state for subsequent branching
                                      because channel will be changed;
                                      SHIFT_DIRECTION_TEMP is a temporary
                                      register*/
LOOP                            /*since a separate pair of CHANNEL_CONTROL
                                  and PIN_CONTROL parameters are associated
                                  with each stepper motor channel, the following
                                  references are to those parameters associated
                                  with the current channel of operation */
    If (SHIFT_DIRECTION_TEMP = 0) then
        Rotate CHANNEL_CONTROL left
    }
    Else {                          /* SHIFT_DIRECTION_TEMP = 1 */
        Rotate CHANNEL_CONTROL right
    }
    If (bit 15 of CHANNEL_CONTROL = 0) then
        Configure pin logic for high to low transition
    }
    Else {                          /* bit 15 of CHANNEL_CONTROL = 1 */
        Configure pin logic for low to high transition
    }
    If (MOD_CNT = 0) then
        CHANNEL_CONTROL = PIN_CONTROL
    }
    Setup next match time = MATCH_TEMP
    Negate all service requests
    If (LAST_SEC_CHAN = current channel) then {
        Change channel to (current channel + 1)
        Goto LOOP
    }
```

The following table shows the SM state transitions listing the service request sources and channel conditions from current state to next state. **Figure 4** illustrates the flow of SM states.

SM State Transition Table

Current State	HSR	M/TSR	LSR	Pin	Flag0	Next State
Any State	10 11	— —	— —	— —	— —	S1 *Init* S2 *Step_Request*
S1 *Init*	00 00	1 1	— —	x x	1 0	S4 *Right_Rot_Chn_Cntl* S3 *Left_Rot_Chn_Cntl*
S2 *Step_Request*	00 00	1 1	— —	x x	1 0	S4 *Right_Rot_Chn_Cntl* S3 *Left_Rot_Chn_Cntl*
S3 *Left_Rot_Chn_Cntl*	00 00	1 1	— —	x x	1 0	S4 *Right_Rot_Chn_Cntl* S3 *Left_Rot_Chn_Cntl*
S4 *Right_Rot_Chn_Cntl*	00 00	1 1	— —	x x	1 0	S4 *Right_Rot_Chn_Cntl* S3 *Left_Rot_Chn_Cntl*
Unimplemented Conditions	01 00	— 0	— 1	— —	— —	— —

NOTES:

1. Conditions not specified are “don't care.”
2. HSR = Host service request
   LSR = Link service request
   M/TSR = Either a match or transition (input capture) service request occurred (M/TSR = 1) or neither occurred (M/TSR = 0).

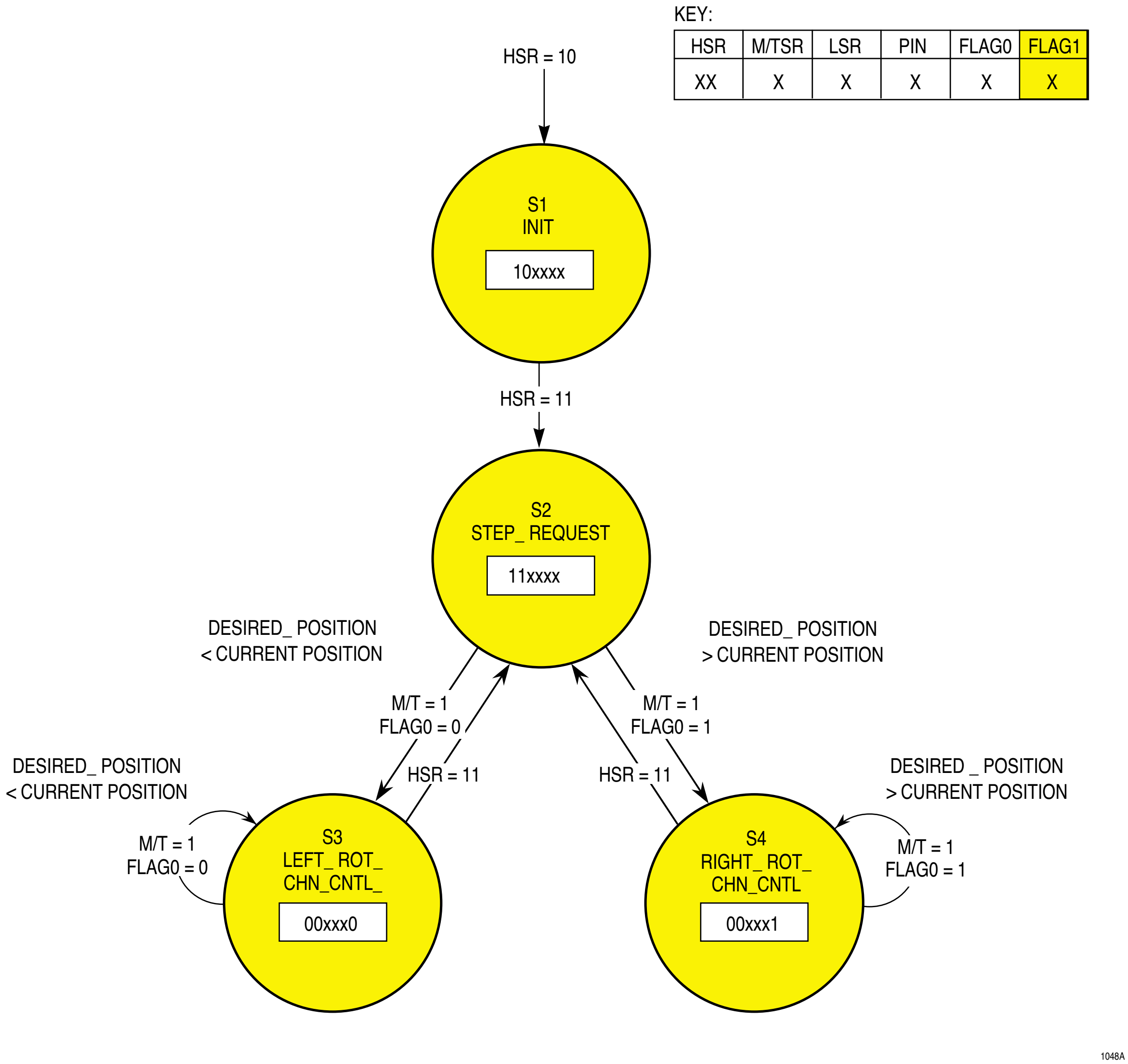


Figure 4 SM State Flow Diagram

NOTES

NOTES

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution;
P.O. Box 5405, Denver Colorado 80217. 1-800-441-2447, (303) 675-2140
Mfax™: RMFAX0@email.sps.mot.com - TOUCHTONE (602) 244-6609, U.S. and Canada Only 1-800-774-1848
INTERNET: http://Design-NET.com
JAPAN: Nippon Motorola Ltd.; Tatsumi-SPD-JLDC,
6F Seibu-Butsuryu-Center, 3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 81-3-3521-8315
ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park,
51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

MOTOROLA