Freescale Semiconductor
PROGRAMMING NOTE

Order this document
by TPUPN04/D

Table Stepper Motor TPU Function (TSM)

by Jeff Wright

1 Functional Overview

The table stepper motor (TSM) function provides the TPU with the capability to drive two-phase stepper motors in full- or half-step modes. The TPU can accelerate the motors, run them at constant speed (or slew), and decelerate them independently of the CPU. The CPU need only initialize the function once and then supply a desired position each time a move is required. The acceleration/deceleration profile is freely configurable by the user via a variable length table that offers up to 82 step rates. The TPU can control up to eight motors in full-step mode, four motors in half-step mode or a combination of both.

2 Detailed Description

The TSM function supports full- and half-step unipolar and bipolar drive of two-phase stepper motors using two or four adjacent TPU channels. Given a move request by the CPU, the TPU independently accelerates, slews and decelerates the motor to the desired position thus relieving the CPU of almost all the overhead associated with controlling the motor. The current motor position is maintained by the TPU as a 16-bit parameter that can be read by the CPU at any time.

The CPU requests a move by writing a 16-bit desired position value and issuing a host service request to the TPU. When the TPU has completed moving the motor to the desired position, it issues an interrupt request to the CPU — if the appropriate interrupt enable bit is set then a CPU interrupt is made, allowing optional interrupt driven control.

The algorithm employed in the TPU re-evaluates the requested destination on every step. This means that the CPU can change the desired position at any time during a movement and the TPU will adjust its strategy to get to the new desired position as quickly as possible. That is, if a motor is moving clockwise from position A to position B at a given slew rate when the CPU writes a new desired position C, which is counterclockwise from the current position, the TPU will immediately decelerate the motor, reverse direction, accelerate, slew, and decelerate to reach position C.

TSM generates the actual step patterns to drive the motor via synchronized output matches on two or four channels. The step patterns generated can be defined by the user. The TSM function operates on a master channel and either one or three slave channels. Except during initialization, all TPU service activity and CPU communication occurs on the master channel only. This keeps TPU loading to a minimum and hence maximizes performance. The master channel is chosen by the user and slaves are then defined as the one or three channels immediately after the master in numeric order. For instance, in two-channel mode, if channel 5 is chosen as the master, channel 6 is the slave. In four-channel mode, if channel 13 is chosen as the master, then channels 14, 15, and 0 are slaves. The choice of two- or four-channel mode is made via a master channel control bit.

The TSM function uses the same user-defined step period profile during acceleration and deceleration. The user specifies this profile in parameter RAM. A 15-bit start period defines the period of the first and last steps in any move, i.e., the start/stop rate (pull-in rate) of the motor. The acceleration profile is programmed into a table of 8-bit constants that are used sequentially to fractionally multiply the start period during acceleration to obtain the 'nth' step period.

The user also specifies a slew period, which defines the exact maximum running speed of the motor. When accelerating, the TPU uses a new value from the acceleration table for each step until the calculated step period (table parameter ∗ start period / 256) is smaller than the slew period. When this point is reached, the TPU switches to the slew period. The TPU also uses the slew period if it reaches the end of the acceleration table. The slew period parameter allows the terminal speed of the motor to be controlled independently of acceleration table length and content.

There are two acceleration table configurations available with the TSM function. These are referred to as local table configuration and split table configuration. Each configuration has different merits and provides different maximum table lengths. The actual table size, within the limits for the various modes explained below, is programmable by the user. Figure 2 illustrates these points.

In local table configuration, acceleration parameters are obtained from a table starting in the lower byte of the first parameter of the first slave channel (the channel immediately after the master in numeric order). Any channel can be a master in local table configuration. Maximum table length is determined by the amount of contiguous parameter RAM available, starting with parameter 0 of the first slave channel. There are four-byte gaps between most of the channels (See Figure 1). If the master is any channel from 0 to 12 or channel 15, then the maximum table size is 12 bytes. If channel 14 is master, maximum table length is 28 bytes; and if channel 13 is master, maximum length is 44 bytes. The maximum number of step rates is equal to table size plus 2 (start/stop and slew). Table length is independent of operating mode. If channel 14 is the master in a two channel configuration, the parameter RAM of channel 0 can still be used to increase the table size to 28 bytes (although channel 0 could not then run another TPU function).

In split table configuration, the acceleration parameter table is split between the parameter RAM of the slave channels and the contiguous block of parameter RAM in channels 14, 15 and 0 (see Figure 1). Since all available slave parameter RAM is used in this configuration, maximum table size is different for two- and four-channel operating modes. When split table configuration is used to control multiple motors, the acceleration parameters in the slave channels for each motor are unique, but the parameters for channels 14, 15 and 0 are shared by all motors. Since start period and slew rate are independently programmable for each motor, motors that share a partially common acceleration table can have different velocity profiles. Since parameter RAM for channels 14, 15 and 0 is used to form the upper part of the acceleration profile, a special case exists when channel 13 is master. When operating in four-channel mode, split table configuration cannot be used with channel 11 or 12 programmed as the master channel unless table length is less than 24 or 12 bytes respectively.

In two-channel mode, split table configuration has the following effects.

> When channel 13 is not the master, acceleration parameters 1 to 12 are obtained from the parameter RAM of the first slave channel, starting with acceleration parameter 1 in the lower byte of parameter word 0.
>
> When channel 13 is the master, acceleration parameters 1 to 12 are obtained from the parameter RAM of channel 2, starting with acceleration parameter 1 in the lower byte of parameter word 0.

In both cases, when table size exceeds 12 parameters the remainder is obtained from the contiguous parameter RAM of channels 14, 15, and 0, starting in the lower byte of parameter word 0 of channel 14. In this configuration the maximum table length including the 12'local' parameters is 56 bytes, giving 58 step rates including start/stop and slew.

In four-channel mode, split table configuration has the following effects.

> When channel 13 is not the master channel, acceleration parameters 1 to 12 are obtained from the parameter RAM of the first slave channel (immediately following the master channel), starting with acceleration parameter 1 in the lower byte of parameter word 0. Acceleration parameters 13 to 24 are obtained from the parameter RAM of the second slave, starting with acceleration parameter 13 in the lower byte of parameter word 0. Acceleration parameters 25 to 36 are obtained from the parameter RAM of the third slave, starting with acceleration parameter 25 in the lower byte of parameter word 0.

When channel 13 is the master channel, acceleration parameters 1 to 12 are obtained from the parameter RAM of channel 2, starting with acceleration parameter 1 in the lower byte of parameter RAM parameter 0. Acceleration parameters 13 to 24 are obtained from the parameter RAM of channel 3, starting with acceleration parameter 13 in the lower byte of parameter word 0. Acceleration parameters 25 to 36 are obtained from the parameter RAM of channel 4, starting with acceleration parameter 25 in the lower byte of parameter word 0.

In both cases, when table size exceeds 36 parameters, the remainder is obtained from the contiguous parameter RAM of channels 14, 15 and 0, starting in the lower byte of parameter word 0 of channel 14. In this configuration, maximum table length including the 36'local' parameters is 80 bytes, giving 82 step rates including start/stop and slew.

See **6 Performance and Use of TSM Function** for more information.

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. TSM function code size is:

97 µ instructions + 8 entries = **105 long words**

4 TSM Function Parameters

This section provides detailed descriptions of function parameters stored in channel parameter RAM. **Figure 1** shows TPU parameter RAM address mapping. Note the contiguous block of parameter RAM in channels 14, 15, and 0 from address $YFFFE0 to $YFFF0A. **Figure 3** shows the parameter RAM assignment used by the function for the various modes of operation. 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	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

	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1	0
$YFFFW0	DESIRED_POSITION		
$YFFFW2	▒ CURRENT_POSITION		
$YFFFW4	TABLE_SIZE	▒ TABLE_INDEX	
$YFFFW6	▒ SLEW_PERIOD		▒ S
$YFFFW8	START_PERIOD		A
$YFFFWA	▒ PIN_SEQUENCE		
$YFFFWC	■		
$YFFFWE	■		

W = Channel number

Parameter Write Access

⬜	Written by CPU
░	Written by TPU
▒	Written by CPU and TPU
■	Unused parameters

Figure 2 a Master Channel Parameter Assignment —All Modes

	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
$YFFF(W+1)0	ACCEL_RATIO_2	ACCEL_RATIO_1
$YFFF(W+1)2	ACCEL_RATIO_4	ACCEL_RATIO_3
$YFFF(W+1)4	ACCEL_RATIO_6	ACCEL_RATIO_5
$YFFF(W+1)6	ACCEL_RATIO_8	ACCEL_RATIO_7
$YFFF(W+1)8	ACCEL_RATIO_10	ACCEL_RATIO_9
$YFFF(W+1)A	ACCEL_RATIO_12	ACCEL_RATIO_11

The Parameters Below Are Not Available In All Cases —See Main Text For Details

$YFFF(W+1)C	ACCEL_RATIO_14	ACCEL_RATIO_13
$YFFF(W+1)E	ACCEL_RATIO_16	ACCEL_RATIO_15
$YFFF(W+2)0	ACCEL_RATIO_18	ACCEL_RATIO_17
$YFFF(W+2)2	ACCEL_RATIO_20	ACCEL_RATIO_19
$YFFF(W+2)4	ACCEL_RATIO_22	ACCEL_RATIO_21
$YFFF(W+2)6	ACCEL_RATIO_24	ACCEL_RATIO_23
$YFFF(W+2)8	ACCEL_RATIO_26	ACCEL_RATIO_25
$YFFF(W+2)A	ACCEL_RATIO_28	ACCEL_RATIO_27
$YFFF(W+2)C	ACCEL_RATIO_30	ACCEL_RATIO_29
$YFFF(W+2)E	ACCEL_RATIO_32	ACCEL_RATIO_31
$YFFF(W+3)0	ACCEL_RATIO_34	ACCEL_RATIO_33
$YFFF(W+3)2	ACCEL_RATIO_36	ACCEL_RATIO_35
$YFFF(W+3)4	ACCEL_RATIO_38	ACCEL_RATIO_37
$YFFF(W+3)6	ACCEL_RATIO_40	ACCEL_RATIO_39
$YFFF(W+3)8	ACCEL_RATIO_42	ACCEL_RATIO_41
$YFFF(W+3)A	ACCEL_RATIO_44	ACCEL_RATIO_43

W = Master Channel number

Figure 2 b Acceleration Parameter Table
Local Table Configuration
Slave Channel Parameter RAM

	15–8	7–0
$YFFF(W+1)0	ACCEL_RATIO_2	ACCEL_RATIO_1
$YFFF(W+1)2	ACCEL_RATIO_4	ACCEL_RATIO_3
$YFFF(W+1)4	ACCEL_RATIO_6	ACCEL_RATIO_5
$YFFF(W+1)6	ACCEL_RATIO_8	ACCEL_RATIO_7
$YFFF(W+1)8	ACCEL_RATIO_10	ACCEL_RATIO_9
$YFFF(W+1)A	ACCEL_RATIO_12	ACCEL_RATIO_11
$YFFFE0	ACCEL_RATIO_14	ACCEL_RATIO_13
$YFFFE2	ACCEL_RATIO_16	ACCEL_RATIO_15
$YFFFE4	ACCEL_RATIO_18	ACCEL_RATIO_17
$YFFFE6	ACCEL_RATIO_20	ACCEL_RATIO_19
$YFFFE8	ACCEL_RATIO_22	ACCEL_RATIO_21
$YFFFEA	ACCEL_RATIO_24	ACCEL_RATIO_23
$YFFFEC	ACCEL_RATIO_26	ACCEL_RATIO_25
$YFFFEE	ACCEL_RATIO_28	ACCEL_RATIO_27
$YFFFF0	ACCEL_RATIO_30	ACCEL_RATIO_29
$YFFFF2	ACCEL_RATIO_32	ACCEL_RATIO_31
$YFFFF4	ACCEL_RATIO_34	ACCEL_RATIO_33
$YFFFF6	ACCEL_RATIO_36	ACCEL_RATIO_35
$YFFFF8	ACCEL_RATIO_38	ACCEL_RATIO_37
$YFFFFA	ACCEL_RATIO_40	ACCEL_RATIO_39
$YFFFFC	ACCEL_RATIO_42	ACCEL_RATIO_41
$YFFFFE	ACCEL_RATIO_44	ACCEL_RATIO_43
$YFFF00	ACCEL_RATIO_46	ACCEL_RATIO_45
$YFFF02	ACCEL_RATIO_48	ACCEL_RATIO_47
$YFFF04	ACCEL_RATIO_50	ACCEL_RATIO_49
$YFFF06	ACCEL_RATIO_52	ACCEL_RATIO_51
$YFFF08	ACCEL_RATIO_54	ACCEL_RATIO_53
$YFFF0A	ACCEL_RATIO_56	ACCEL_RATIO_55

W = Master channel number or one if channel 13 is master

Figure 2 c Acceleration Parameter Table
Split Table Configuration (2-Channel Mode)
Slave Channel Parameter RAM Plus Channels 14, 15, and 0

	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
$YFFF(W+1)0	ACCEL_RATIO_2	ACCEL_RATIO_1
$YFFF(W+1)2	ACCEL_RATIO_4	ACCEL_RATIO_3
$YFFF(W+1)4	ACCEL_RATIO_6	ACCEL_RATIO_5
$YFFF(W+1)6	ACCEL_RATIO_8	ACCEL_RATIO_7
$YFFF(W+1)8	ACCEL_RATIO_10	ACCEL_RATIO_9
$YFFF(W+1)A	ACCEL_RATIO_12	ACCEL_RATIO_11
$YFFF(W+2)0	ACCEL_RATIO_14	ACCEL_RATIO_13
$YFFF(W+2)2	ACCEL_RATIO_16	ACCEL_RATIO_15
$YFFF(W+2)4	ACCEL_RATIO_18	ACCEL_RATIO_17
$YFFF(W+2)6	ACCEL_RATIO_20	ACCEL_RATIO_19
$YFFF(W+2)8	ACCEL_RATIO_22	ACCEL_RATIO_21
$YFFF(W+2)A	ACCEL_RATIO_24	ACCEL_RATIO_23
$YFFF(W+3)0	ACCEL_RATIO_26	ACCEL_RATIO_25
$YFFF(W+3)2	ACCEL_RATIO_28	ACCEL_RATIO_27
$YFFF(W+3)4	ACCEL_RATIO_30	ACCEL_RATIO_29
$YFFF(W+3)6	ACCEL_RATIO_32	ACCEL_RATIO_31
$YFFF(W+3)8	ACCEL_RATIO_34	ACCEL_RATIO_33
$YFFF(W+3)A	ACCEL_RATIO_36	ACCEL_RATIO_35
$YFFFE0	ACCEL_RATIO_38	ACCEL_RATIO_37
$YFFFE2	ACCEL_RATIO_40	ACCEL_RATIO_39
$YFFFE4	ACCEL_RATIO_42	ACCEL_RATIO_41
$YFFFE6	ACCEL_RATIO_44	ACCEL_RATIO_43
$YFFFE8	ACCEL_RATIO_46	ACCEL_RATIO_45
$YFFFEA	ACCEL_RATIO_48	ACCEL_RATIO_47
$YFFFEC	ACCEL_RATIO_50	ACCEL_RATIO_49
$YFFFEE	ACCEL_RATIO_52	ACCEL_RATIO_51
$YFFFF0	ACCEL_RATIO_54	ACCEL_RATIO_53
$YFFFF2	ACCEL_RATIO_56	ACCEL_RATIO_55
$YFFFF4	ACCEL_RATIO_58	ACCEL_RATIO_57
$YFFFF6	ACCEL_RATIO_60	ACCEL_RATIO_59
$YFFFF8	ACCEL_RATIO_62	ACCEL_RATIO_61
$YFFFFA	ACCEL_RATIO_64	ACCEL_RATIO_63
$YFFFFC	ACCEL_RATIO_66	ACCEL_RATIO_65
$YFFFFE	ACCEL_RATIO_68	ACCEL_RATIO_67
$YFFF00	ACCEL_RATIO_70	ACCEL_RATIO_69
$YFFF02	ACCEL_RATIO_72	ACCEL_RATIO_71
$YFFF04	ACCEL_RATIO_74	ACCEL_RATIO_73
$YFFF06	ACCEL_RATIO_76	ACCEL_RATIO_75
$YFFF08	ACCEL_RATIO_78	ACCEL_RATIO_77
$YFFF0A	ACCEL_RATIO_80	ACCEL_RATIO_79

W = Master channel number or 1 if channel 13 is master

Figure 2 d Acceleration Parameter Table
Split Table Configuration (4-Channel Mode)
Slave Channel Parameter RAM Plus Channels 14, 15, and 0

4.1 DESIRED_POSITION

This 16-bit parameter contains the desired position (destination) of the stepper motor. The CPU can write DESIRED_POSITION at any time. If the motor is not already moving then a host service request (HSR) type %11 must be issued to the master channel to initiate the move. The range for DESIRED_POSITION is $0000 to $FFFF.

4.2 CURRENT_POSITION

This 16-bit parameter is maintained by the TPU. It contains the current position of the stepper motor. The parameter is incremented or decremented for each completed step depending on the direction of the step. In this way CURRENT_POSITION tracks the movement of the motor. The motor stops when it has decelerated to the start/stop rate and CURRENT_POSITION is equal to DESIRED_POSITION. CURRENT_POSITION is updated after the relevant step has completed, but the exact timing of the update cannot be predicted due to the service scheme of the TPU. For this reason when CURRENT_POSITION is read while the motor is moving there can be an error of ±1 step. After the TPU has issued the interrupt request at the end of the move, CURRENT_POSITION will be accurate.

CURRENT_POSITION should be initialized by the CPU as part of the function initialization.

4.3 TABLE _SIZE

This 8-bit parameter, initialized by the CPU, defines the length of the acceleration table. The valid range for TABLE_SIZE is one to maximum. In local table configuration, maximum is 12 when channels 0 to 12 or 15 are specified as the master channel. Maximum is 44 if channel 13 is chosen as the master channel and 28 if channel 14 is chosen as the master. In local/remote split table configuration, maximum table size is 56 in two-channel mode and 80 in four-channel mode. Maximum table size is reduced if channel 14, 15, or 0 is programmed as a TSM master channel or to run another TPU function. TABLE_SIZE should not be written while the motor is moving.

4.4 TABLE _INDEX

This 8-bit parameter is used by the TPU as a pointer into the acceleration parameter table. Update timing is not specified for TABLE_INDEX and it is not recommended for interpretation by the user. TABLE_INDEX must be written to zero by the CPU during initialization and then never written again. Writing TABLE_INDEX while the motor is running will result in indeterminate operation.

4.5 BIT_S

This bit flag is used internally by the TPU to track slew rate operation. Update timing is not specified for BIT_S and it is not recommended for interpretation by the user. BIT_S must be written to zero by the CPU during initialization and then never written again.

4.6 SLEW_PERIOD

This 15-bit parameter is written by the CPU. It determines the slew rate (maximum stepping speed) of the stepper motor. The value programmed into SLEW_PERIOD determines the step period in TCR1 clocks during the constant speed part of a move. The valid range for SLEW_PERIOD is from one to START_PERIOD, although in practice the minimum sustainable SLEW_PERIOD is determined by TPU latency and motor characteristics. See **6 Performance and Use of TSM Function** for more details. SLEW_PERIOD should not be changed while the motor is moving.

4.7 BIT_A

This control bit determines whether two or four TPU channels are used by the TSM function. If BIT_A = 0 then two channels are used (master plus one slave) and if BIT_A = 1 then four channels are used (master plus three slaves). BIT_A selection is determined by the mode of stepping and driving: full- or half-step and unipolar or bipolar drive. The slave channels always follow the master channel in numeric order — if channel 1 is selected as master then channel 2 or channels 2, 3, and 4 are used by TSM as slaves. BIT_A must be initialized by the CPU prior to issuing the first move request HSR. BIT_A should not be changed while the function is running.

4.8 START_PERIOD

This 15-bit parameter is written by the CPU. It determines the start/stop rate (pull-in rate) of the stepper motor. The value programmed into START_PERIOD determines the step period in TCR1 clocks during the first and last steps of a move. The value is also used as a base value in the calculation to determine the step periods in the acceleration/deceleration phases. The valid range for START_PERIOD is from 1 to $7FFF, although in practice the minimum sustainable START_PERIOD is determined by TPU latency and motor characteristics. Refer to **6 Performance and Use of TSM Function** for more details. START_PERIOD should not be changed while the motor is moving.

4.9 PIN_SEQUENCE

This 16-bit parameter, along with host sequence bit 1 (HSQ1) determines the step patterns that are produced on the two or four TPU pins during stepping. It is initialized by the CPU to contain the sequence of pin levels required on the master TSM channel. To generate a step, PIN_SEQUENCE is rotated left or right once, depending on step direction and the pin level that will result after the step match is determined by the MSB of the new PIN_SEQUENCE. Example values for full-step two-channel mode and half-step four-channel mode are $3333 and $E0E0 respectively.

The pin responses of slave channels are also determined by PIN_SEQUENCE. Given the sequence of pin levels for one channel, the sequence for the other channels can be derived by right rotating the same sequence either once or twice between each slave and using the resulting MSB. The choice of one or two rotates of PIN_SEQUENCE between slaves is made with HSQ1. See **6 Performance and Use of TSM Function** for more details.

The channel initialize host service request (HSR %01 or %10) issued to each of the TSM channels should correspond to the initial PIN_SEQUENCE value. Refer to **4 TSM Function Parameters** for more details.

4.10 ACCEL_RATIO_1.... ACCEL_RATIO_N

These 8-bit parameters make up the acceleration parameter table. The CPU initializes table parameters. The START_PERIOD parameter is fractionally multiplied by successive table values to determine step periods during acceleration and deceleration phases. Step period is equal to:

$$\text{STEP\_PERIOD} = (\text{START\_PERIOD} * \text{ACCEL\_RATIO\_N})/256$$

The resulting value is in TCR1 clocks. ACCEL_RATIO_1 has a valid range of $01 to $FF. These parameters should not be changed while the motor is stepping.

4.11 HSQ0

Host sequence bit 0 on the master channel is used to select the type of acceleration parameter table. If HSQ0 = 0, then the local table configuration is selected. If HSQ0 = 1, the split table configuration is selected. HSQ0 should be initialized by the CPU prior to issuing the first move request HSR and should not be changed while the motor is running.

4.12 HSQ1

Host sequence bit 1 on the master channel is used to select the number of rotates of PIN_SEQUENCE between slave channels. If HSQ1 = 0, one rotate is performed. If HSQ1 = 1, two rotates are performed. See **6 Performance and Use of TSM Function** for more information. HSQ1 must be initialized by the CPU prior to issuing the first move request HSR and should not be changed while the motor is running.

5 Host Interface to Function

This section provides information concerning the TPU host interface to the function. **Figure 4** 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 (MM) bit 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] — 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	

CH[15:0]	Operating Mode — Only Used For Master Channel
X0	Local acceleration table
X1	Split acceleration table
0X	Rotate PIN_SEQUENCE once between slave channels
1X	Rotate PIN_SEQUENCE twice between slave channels

HSRR[0:1] — 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[15:0]	Initialization
00	No Host Service (Reset Condition)
01	Initialize, Pin low
10	Initialize, Pin high
11	Move Request (master only)

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[15:0]	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

5.1 TSM Function Configuration

The CPU configures the TSM function as follows. For configuration of the overall operation of the TPU module, such as prescaler selection etc., refer to the *TPU Reference Manual* (TPURM/AD).

1. The appropriate channel priority bits are cleared, disabling the master and slave TSM channels.
2. The TSM function number is written to the channel function select bits of both the master channel and slave channels.
3. The interrupt control registers are initialized if the function is to be interrupt driven.
4. An acceleration table is written by the CPU into TPU parameter RAM.
5. DESIRED_POSITION and CURRENT_POSITION are both initialized to the same value.
6. TABLE_SIZE is written to reflect acceleration table size and TABLE_INDEX is written to $00.
7. SLEW_PERIOD, START_PERIOD, BIT_S, and BIT_A are written with SLEW_PERIOD < START_PERIOD and BIT_A determining mode of operation. BIT_S should be cleared.
8. PIN_SEQUENCE is written with a value that will determine the channel pin responses as the motor steps. Bit 14 or 0 of PIN_SEQUENCE determines the pin state of the master channel after the first step match depending on the direction of the first step.
9. The host sequence bits, HSQ[1:0] of the master channel are written to select operating mode.
10. An HSR %01 or %10 is issued to each TSM channel to initialize the function and set the channel pin to the desired initial output state. The HSR issued to the master channel should match the MSB of PIN_SEQUENCE and the HSR issued to the slaves should match their corresponding bit in PIN_SEQUENCE.

Example 1: four-channel mode with two rotates of PIN_SEQUENCE between channels:

If PIN_SEQUENCE = $E0E0, then the following HSRs should be issued

Master	HSR %10 (pin high)
Slave 1	HSR %01 (pin low)
Slave 2	HSR %01 (pin low)
Slave 3	HSR %10 (pin high)

Example 2: two-channel mode with one rotate of PIN_SEQUENCE between channels:

If PIN_SEQUENCE = $9999, then the following HSRs should be issued

Master	HSR %10 (pin high)
Slave 1	HSR %10 (pin high)

11. The channel priority bits are written to enable the function and assign channel priority.
12. The TPU executes the selected initialization states.

After each channel has been initialized, the TPU clears the host service request bits and asserts an interrupt request from that channel. If the channel interrupt enable bit is set then a CPU interrupt will result. When all host service request bits have been cleared by the TPU and/or an interrupt request has been generated from all the TSM channels, the CPU can assume that the motor is correctly initialized.

Once initialization is complete, the CPU controls the TSM function through the master channel only. The CPU can now issue a move request to the TSM function in the following manner:

1. Writing the required motor position to DESIRED_POSITION.
2. Issuing a move request host service request to the master channel (HSR = %11).

The TPU will then accelerate, slew and decelerate the motor to the desired position, issuing an interrupt request to the CPU when the move is complete. If the master channel interrupt enable bit is set, then a CPU interrupt will result. The CPU can issue a move request in this manner at any time, even while the motor is still moving. In this case the current step is completed and the TPU then adjusts its strategy to move the motor to the new DESIRED_POSITION as quickly as possible, even if this involves decelerating and reversing direction — see later examples.

6 Performance and Use of TSM Function

6.1 Performance

Like all TPU functions, TSM 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 the TPU is driving a single stepper motor using TSM in two-channel mode, and no other TPU channels are active, the minimum step period is 186 CPU clocks. This is approximately equivalent to 90,000 pulses per second at 16.77 MHz bus speed and 114,000 pulses per second at 20.97 MHz bus speed. In four-channel mode the equivalent figures are 234 CPU clocks, 71,000 pulses per second at 16.77 MHz bus and 89,000 pulses per second at 20.97 MHz. When more TPU functions are active or multiple motors are implemented, performance decreases — e.g., if two motors were driven in two-channel mode (four active TPU channels) then the maximum pulse rate for each motor would be approximately half that given above. 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 the information in the TSM state timing table below.

Table 1 Table Stepper Motor Function — State Timing

State Number and Name	Max. CPU Clock Cycles	RAM Accesses by TPU
S1 TSM_INIT_LO	6	1
S2 TSM_INIT_HI	6	1
S3 TSM_MOVE_REQ 2 Channel mode 4 Channel mode Already Stepping	 162 210 6	 17 17 1
S4 TSM_STEP_MATCH 2 Channel mode 4 Channel mode	 172 220	 20 20

NOTE: Execution times do not include the time slot transition time (TST = 10 or 14 CPU clocks)

6.2 Generating Step Patterns

The TSM function has been designed to provide as much flexibility as possible in the generation of the step patterns that drive the motor. Any value can be written into the PIN_SEQUENCE parameter and the choice of one or two rotates of PIN_SEQUENCE between channels increases the flexibility further. This flexibility may allow the TSM function to meet the needs of an unusual drive scheme. However, since the primary purpose of the TSM function is to drive stepper motors in a conventional manner, it has been tested using the two stepping schemes described below:

In full-step two-channel mode (see **Figure 4**), the PIN_SEQUENCE parameter should be initialized to $3333 or a shifted version of this value such as $6666 or $9999. Master channel host sequence bit 1 should be cleared to select one rotate of PIN_SEQUENCE between channels. The initial value of PIN_SEQUENCE written to parameter RAM defines the starting point of the step sequence.

To generate a step, the PIN_SEQUENCE is rotated left or right once, depending on the motor direction. Master channel pin level at the end of the step (i.e. when the next match occurs) is defined by the MSB of the rotated PIN_SEQUENCE. The new PIN_SEQUENCE value is stored in parameter RAM. The pin level of the slave channel is obtained by further rotating a copy of the new PIN_SEQUENCE right once. The value of the resulting MSB determines the slave pin level. **Figure 4** shows the effective positions of the bits that determine the pin levels of the master and slave channels.

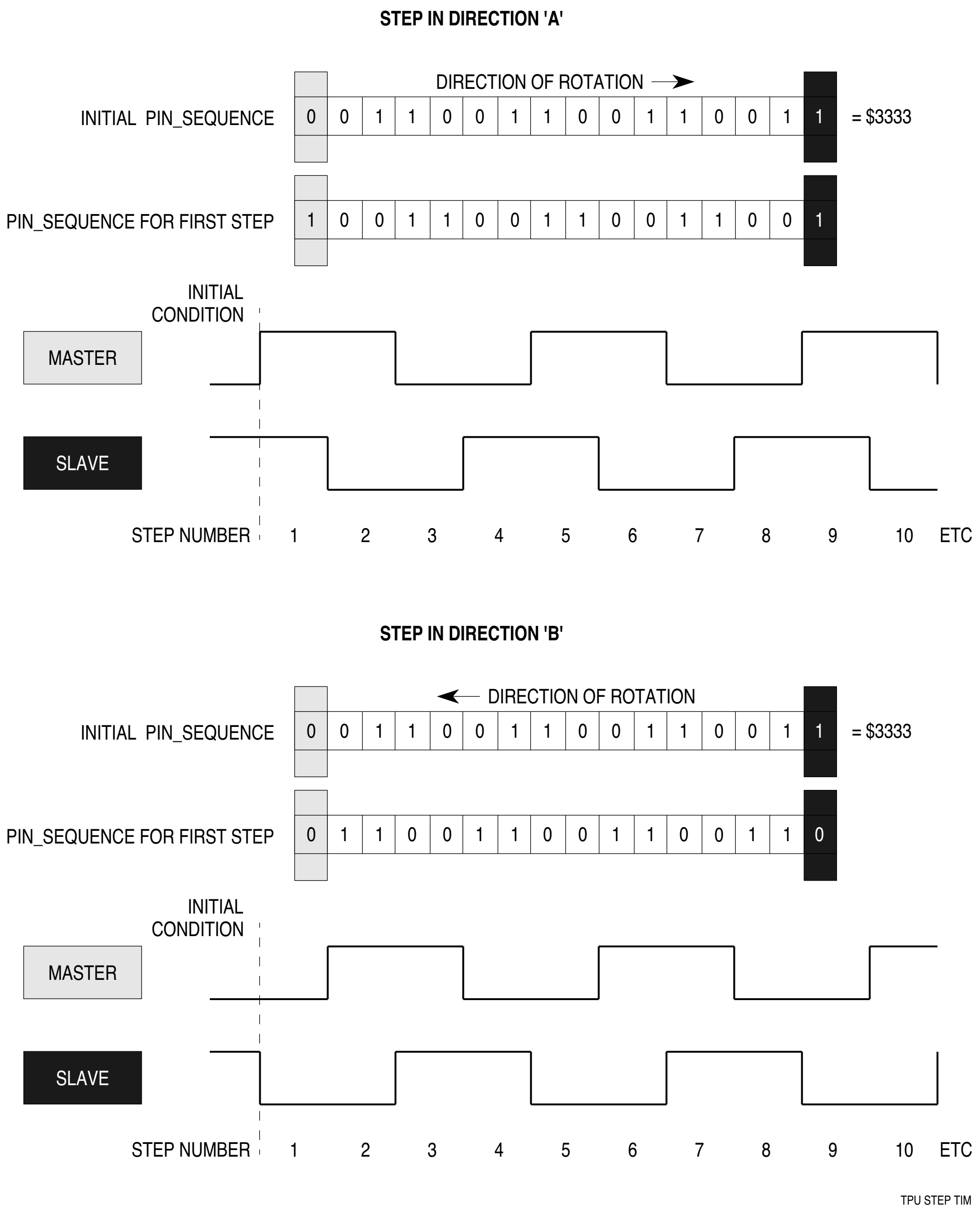


Figure 4 Two-Channel Mode —Full-Step Generation

During initialization an HSR request is issued to each TSM channel to configure the initial pin level. The HSR type issued to each channel should match the value of the corresponding bit in the initial PIN_SEQUENCE. For example, if $3333 is written to PIN_SEQUENCE, then an HSQ %01 should be issued to the master channel (pin low) and an HSR %10 to the slave channel (pin high).

In half-step four-channel mode (See **Figure 6**), the PIN_SEQUENCE parameter should be initialized to $E0E0 or a shifted version of this value such as $C1C1 or $8383. Master channel host sequence bit 1 should be set to select two rotates of PIN_SEQUENCE between channels. The initial value of PIN_SEQUENCE written to parameter RAM defines the starting point of the step sequence.

To generate a step, the PIN_SEQUENCE is rotated left or right once, depending on the motor direction. Master channel pin level at the end of the step (i.e. when the next match occurs) is defined by the MSB of the rotated PIN_SEQUENCE. The new PIN_SEQUENCE value is stored in parameter RAM. The pin level of the first slave channel is obtained by further rotating a copy of the new PIN_SEQUENCE right twice. The value of the resulting MSB determines slave 1 pin level. Similarly, the pin levels of the first and second slaves are determined by the MSB after further right-rotating the copy of PIN_SEQUENCE two times and four times, respectively. **Figure 6** shows the effective positions of the bits that determine the pin levels of the master and slave channels.

During initialization an HSR request is issued to each TSM channel to configure the initial pin level. The HSR type issued to each channel should match the value of the corresponding bit in the initial PIN_SEQUENCE. For example, if $E0E0 is written to PIN_SEQUENCE then an HSR %01 should be issued to slaves 1 and 2 (pin low) and an HSR %10 to the master and slave 3 (pin high).

6.3 TSM Positioning Algorithm

This section is designed to give an overview of the positioning algorithm employed by the TSM function. It provides all the detail necessary to understand normal function and use of TSM. For more detail on the microcode operation, refer to the pseudocode in **8 TSM Function Algorithm**.

6.4 Simple A to B Move Request

When the CPU makes a move request, the TSM function first checks to see if the motor is stepping. If it is, no further action is taken until the next step match is serviced. If the motor is not stepping, DESIRED_POSITION is checked against CURRENT_POSITION. If the two values are the same, no steps are generated and an interrupt request is issued to the CPU. If CURRENT_POSITION does not equal DESIRED_POSITION, the algorithm uses the following test to determine in which direction to step the motor.

TEMP = DESIRED_POSITION – CURRENT_POSITION

If TEMP MSB = 1, then step in direction B (rotate PIN_SEQUENCE left for step generation)
Else step in direction A (rotate PIN_SEQUENCE right for step generation)

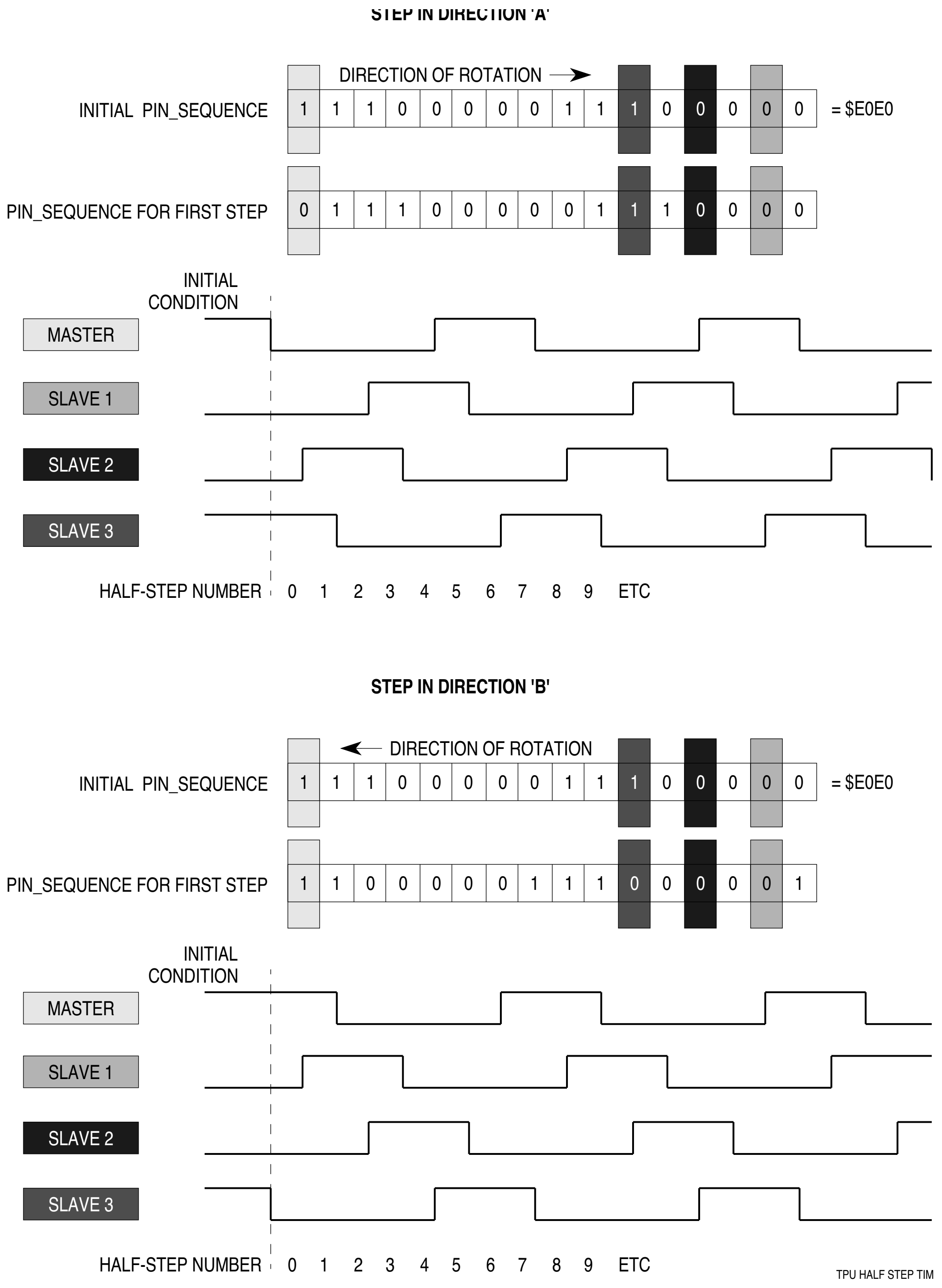


Figure 5 Four-Channel Mode — Half-Step Generation

The TSM function then generates steps in the required direction. The first step has a period equal to START_PERIOD TCR1 clocks. After the first step, the function accelerates the motor using step periods derived from the table and START_PERIOD. If the derived period is less than SLEW_PERIOD then SLEW_PERIOD is used instead of the table derived period. The function continues to accelerate as long as there are sufficient steps left in the move to decelerate back to the start period before reaching DESIRED_POSITION. Examples of this process for short moves are shown in **Figure 6**. Note that two steps are taken with period equal to START_PERIOD at the end of each move. If the end of the acceleration table is reached, the next and subsequent steps until deceleration are generated with period equal to SLEW_PERIOD.

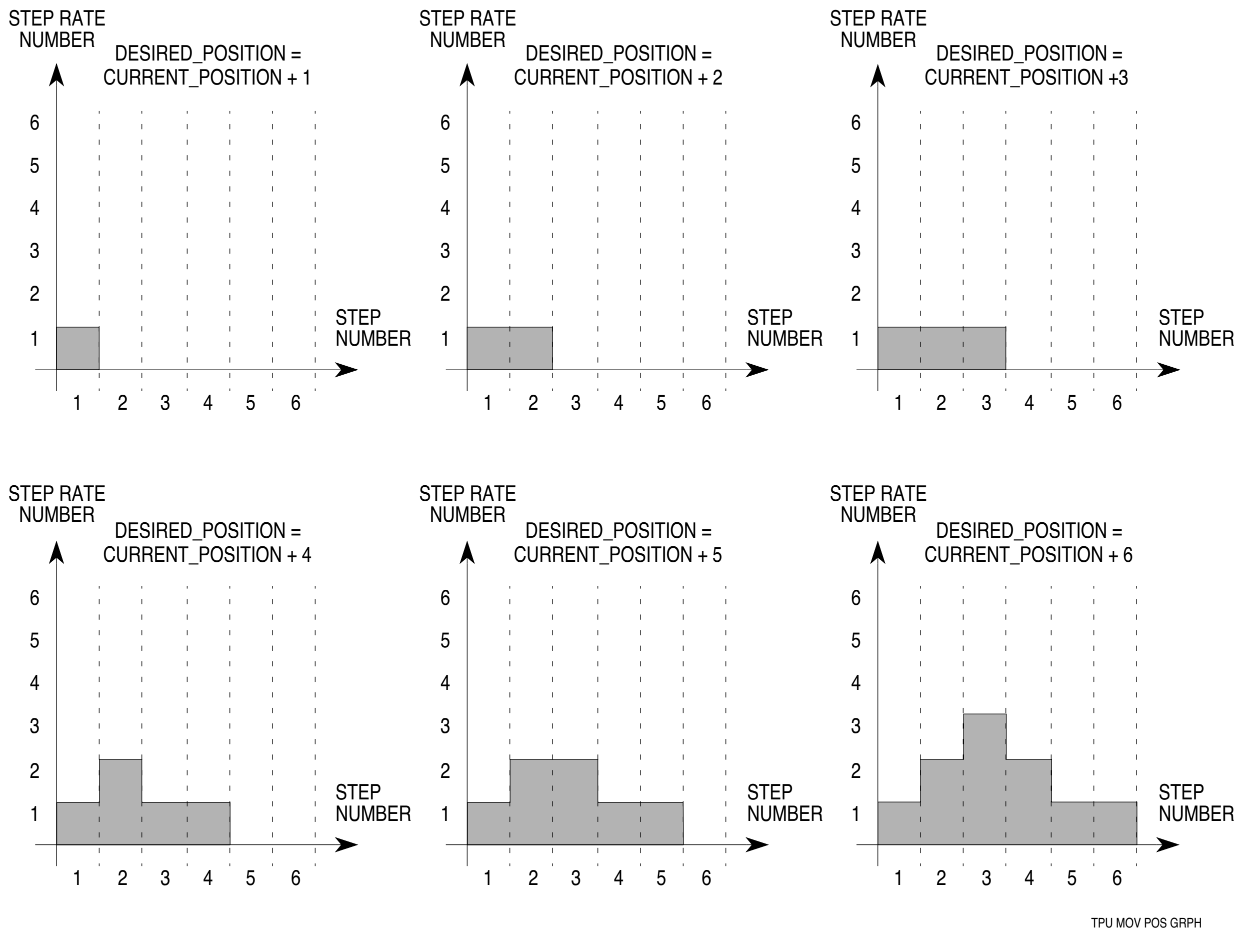


Figure 6 Short Move Position Algorithm Examples
No Change in DESIRED_POSITION in Mid-Move

6.5 Changing DESIRED_POSITION in Mid-Move

Because DESIRED_POSITION is re-evaluated against CURRENT_POSITION after every step, DESIRED_POSITION can be changed by the CPU at any time during a move. This feature is particularly important for fast servo systems. A move request HSR should always be issued when DESIRED_POSITION is changed. There are four possible cases for a changed DESIRED_POSITION value — the TSM function responds differently in each case:

Case 1.New DESIRED_POSITION is in same direction as old DESIRED_POSITION but further away (See **Figure 8a**).

If the motor is slewing, this phase of the move is extended until it is time to decelerate to the new DESIRED_POSITION.

If the motor has started to decelerate, the new DESIRED_POSITION may result in additional acceleration and further slew phase.

Case 2. New DESIRED_POSITION is in same direction as old DESIRED_POSITION but closer (See **Figure 8b**).

If the motor is not closer to the new DESIRED_POSITION than the number of acceleration steps already taken, the motor continues to accelerate and slew until it is time to decelerate to the new DESIRED_POSITION.

If the motor is closer to the new DESIRED_POSITION than the number of acceleration steps already taken, the motor immediately decelerates. This causes overshoot, and the motor will subsequently reverse direction and accelerate (if possible) to the new DESIRED_POSITION.

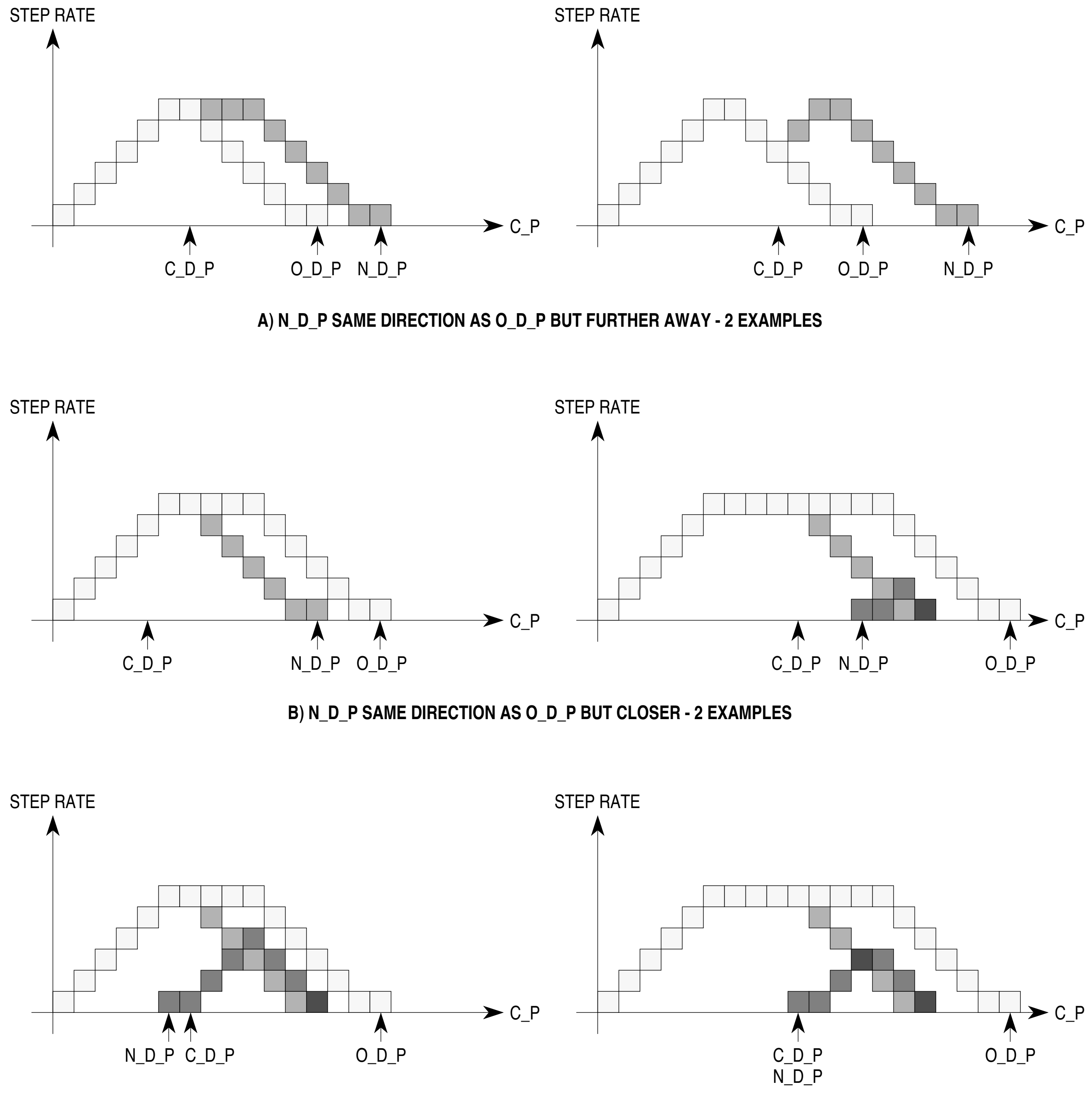
Case 3. New DESIRED_POSITION is in opposite direction to CURRENT_POSITION than old DESIRED_POSITION (See **Figure 8c**).

The motor immediately decelerates, and when it reaches START_PERIOD, it reverses direction and accelerates/slews/decelerates to the new DESIRED_POSITION.

Case 4. New DESIRED_POSITION is the same as CURRENT_POSITION (See **Figure 8c**).

If the last step had a period equal to START_PERIOD, the function stops and issues an interrupt request.

If the last step was not at the start/stop rate, the motor immediately decelerates. This causes overshoot, and the motor will subsequently reverse direction and accelerate (if possible) to the new DESIRED_POSITION.



A) N_D_P SAME DIRECTION AS O_D_P BUT FURTHER AWAY - 2 EXAMPLES

B) N_D_P SAME DIRECTION AS O_D_P BUT CLOSER - 2 EXAMPLES

C) N_D_P OPPOSITE DIRECTION TO O_D_P

D) N_D_P SAME AS C_P

KEY:

C_P = CURRENT_POSITION

O_D_P = ORIGINAL_DESIRED_POSITION

C_D_P = POINT WHERE DESIRED_POSITION IS CHANGED FROM O_D_P TO N_D_P

N_D_P = NEW DESIRED_POSITION

□ = ORIGINAL MOVE IF DESIRED_POSITION NOT CHANGED

▨ = STEP IN DIRECTION **A** AS A RESULT OF CHANGE

▩ = STEP IN DIRECTION **B** AS A RESULT OF CHANGE

■ = STEP MADE IN BOTH DIRECTIONS AS A RESULT OF CHANGE

TPU DELTA POS GRPH

Figure 7 The Effect of Changing DESIRED_POSITION During Mid-Move

6.6 Use of the SLEW_PERIOD Parameter

The slew period parameter allows the minimum step period of the motor (and therefore its terminal speed) to be specified exactly in TCR1 counts, independently of the values in the acceleration table and the value of START_PERIOD. The SLEW_PERIOD parameter is used under two circumstances:

1. When the end of the acceleration table is reached.
2. When the period value obtained from the fractional multiply of the START_PERIOD value by an acceleration parameter from the table is less than SLEW_PERIOD. This allows SLEW_PERIOD to be used to limit the maximum speed of a particular motor when multiple motors are sharing a common acceleration table.

SLEW_PERIOD also allows a motor to make moves of the same length at different speeds without requiring reprogramming of the acceleration table. Figure 8 shows an example of using SLEW_PERIOD for this purpose.

Note that SLEW_PERIOD should only be changed between moves and not while the motor is running.

6.7 Choosing a Table Configuration

The TPU has a limited amount of parameter RAM for table data space and the unimplemented RAM ‘gaps' between channels complicate usage. The local and split table configuration options allow a programmer to use the available space efficiently. Preferred table configuration depends upon how many motors are being controlled and upon how many TPU channels are required for other functions. The following discussions assume that an application requires the maximum possible number of acceleration steps. If this is not the case, a better arrangement may be possible.

6.7.1 Controlling a Single Motor

Split table configuration should be employed when there are spare TPU channels in the application. Configure the TPU so that the spare channels start at channel 14. In this way, the acceleration table can be maximized by using all the slave channel parameter RAM plus the parameter RAM of up to three of the spare channels (channels 14, 15 and 0).

Local table configuration should be employed when there are no spare TPU channels. It is recommended that channel 13 be chosen as the master to provide the largest possible table in the parameter RAM of channels 14, 15 and 0.

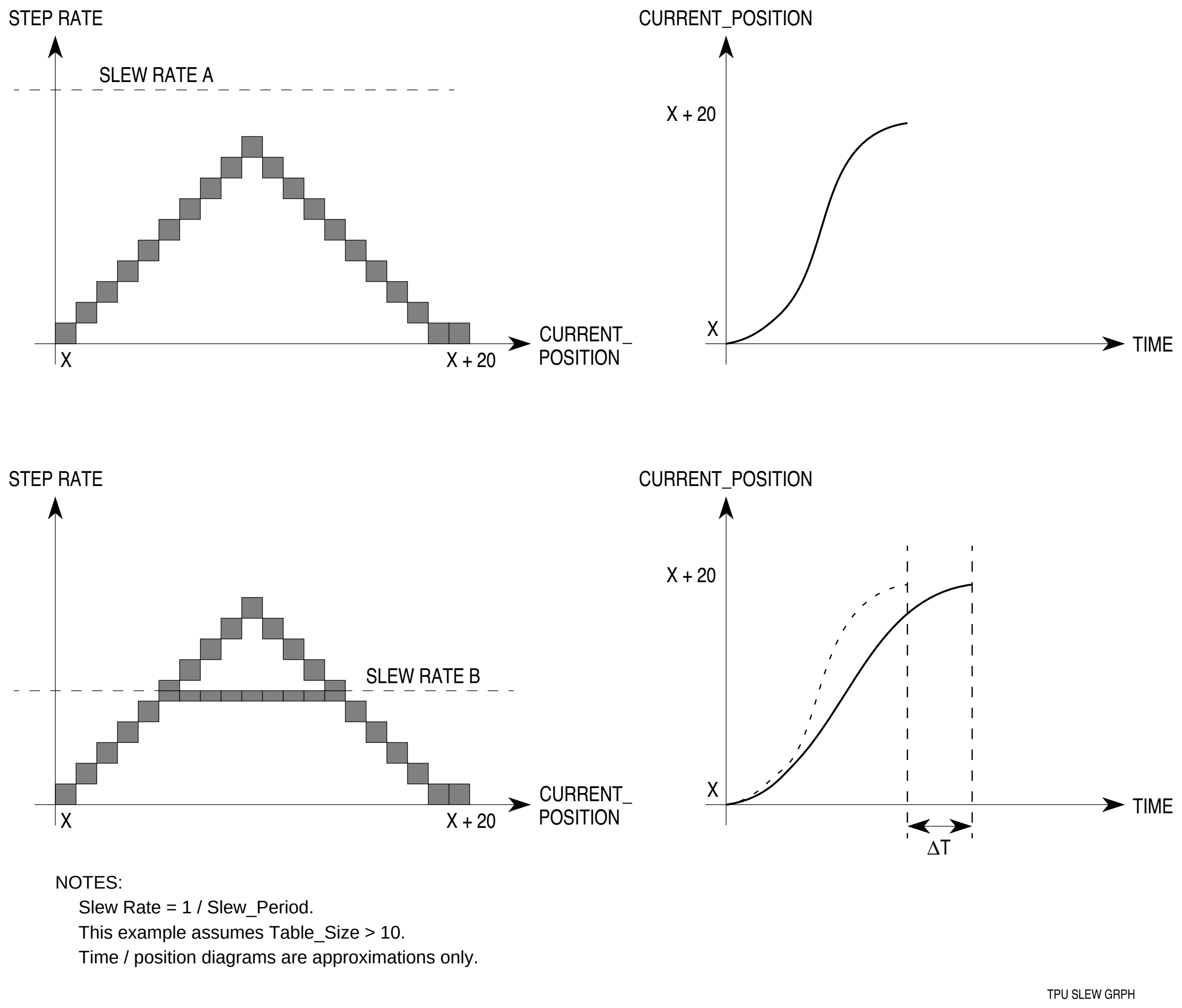


Figure 8 Example of SLEW_PERIOD Effect On a 20-Step Move

6.7.2 Controlling Multiple Motors

In this situation, choice of table configuration is more complex and application dependent.

The local table configuration allows complete individual control in multiple motor configurations, but at the expense of fewer acceleration steps per motor. The motor requiring the finest control should have channel 13 chosen as its master to maximize the acceleration table length.

Many stepper motors can be driven with a step rate profile of the same shape even though individual rates are quite different. There are also many applications that drive multiple motors of identical type. The split table configuration allows multiple motors to share part of an acceleration profile. Using a shared common table for part of the profile allows each of the individual motors to have a larger overall acceleration table. The first 12 or 36 (two- or four-channel mode respectively) parameters are contained in the slave channels, the remainder reside in the parameter RAM of channels 14, 15 and 0 and are shared. Note that channels 14, 15 and 0 can still be used as slaves for a motor programmed to have its master as channel 13.

Several features of the TSM function and the split table configuration make table sharing possible:

The acceleration table only defines the shape of the profile. Actual step rates are derived from START_PERIOD, which is individually programmable for each motor.

Table length is programmable for each motor, so all need not use the full table.

The SLEW_RATE parameter is individually programmable for each motor. The parameter can be used to define different terminal speeds for motors using the same number of steps in the table.

The first 12 or 36 acceleration steps are unique to each motor.

Even in cases where the variety of motor types or usage rules out sharing a table, when four channels are used to drive the motor, split table configuration provides a larger unique table for each motor than a local table can (36 vs. 12), except when channel 13 is the master — then local configuration offers up to 44 acceleration steps.

When channel 13 is programmed as a master, the parameter RAM of its slaves is used for common table entries, and the first 12 or 36 acceleration parameters for this motor must come from somewhere else. In this case the TSM function obtains initial acceleration parameters from channel 2 or channels 2, 3 and 4. In this way, a second motor can be configured with its master as channel 1 and both motors can have access to the same 80-byte acceleration table.

Four example multi-motor configurations that show various channel and parameter RAM assignments follow.

Table 2 Four Motors
Each Controlled by Four Channels (Half-Step)
Split Table Configuration

Channel	Function	Parameter RAM Content
0	Motor 4, Slave 3	Acceleration parameters 69 to 80 for all motors
1	Motor 1, Master	Motor 1 control parameters
2	Motor 1, Slave 1	Acceleration parameters 1 to 12 for motors 1 and 4
3	Motor 1, Slave 2	Acceleration parameters 13 to 24 for motors 1 and 4
4	Motor 1, Slave 3	Acceleration parameters 25 to 36 for motors 1 and 4
5	Motor 2, Master	Motor 2 control parameters
6	Motor 2, Slave 1	Acceleration parameters 1 to 12 for motor 2
7	Motor 2, Slave 2	Acceleration parameters 13 to 24 for motor 2
8	Motor 2, Slave 3	Acceleration parameters 25 to 36 for motor 2
9	Motor 3, Master	Motor 3 control parameters
10	Motor 3, Slave 1	Acceleration parameters 1 to 12 for motor 3
11	Motor 3, Slave 2	Acceleration parameters 13 to 24 for motor 3
12	Motor 3, Slave 3	Acceleration parameters 25 to 36 for motor 3
13	Motor 4, Master	Motor 4 control parameters
14	Motor 4, Slave 1	Acceleration parameters 37 to 52 for all motors
15	Motor 4, Slave 2	Acceleration parameters 53 to 68 for all motors

Table 3 Three Motors
Each Controlled by Two Channels (Full-Step)
Split Table Configuration

Channel	Function	Parameter RAM Content
0*	No function	Acceleration parameters 45 to 56 for all motors
1	Motor 1, Master	Motor 1 control parameters
2	Motor 1, Slave 1	Acceleration parameters 1 to 12 for motors 1 and 3
3	Motor 2, Master	Motor 2 control parameters
4	Motor 2, Slave 1	Acceleration parameters 1 to 12 for motor 2
5	Any other TPU function	—
6	Any other TPU function	—
7	Any other TPU function	—
8	Any other TPU function	—
9	Any other TPU function	—
10	Any other TPU function	—
11	Any other TPU function	—
12	Any other TPU function	—
13	Motor 3, Master	Motor 3 control parameters
14	Motor 3, Slave 1	Acceleration parameters 13 to 28 for all motors
15*	No function	Acceleration parameters 29 to 44 for all motors

*Channels 15 and 0 can be used to run other TPU functions if the acceleration table is programmed to be less than 29 or 45 entries respectively.

Table 4 Three Motors
Motor 1: Two-Channel Mode, 12 Acceleration Parameters
Motor 2: Four-Channel Mode, 30 Acceleration Parameters
Motor 3: Four-Channel Mode, 44 Acceleration Parameters
Local Table Configuration for Motors 1 and 3
Split Table Configuration for Motor 2

Channel	Function	Parameter RAM Content
0	Motor 3, Slave 3	Acceleration parameters 33 to 44 for motor 3
1	Motor 1, Master	Motor 1 control parameters
2	Motor 1, Slave	Acceleration parameters 1 to 12 for motor 1
3	Motor 2, master	Motor 2 control parameters
4	Motor 2, Slave 1	Acceleration parameters 1 to 12 for motor 2
5	Motor 2, Slave 2	Acceleration parameters 13 to 24 for motor 2
6	Motor 2, Slave 3	Acceleration parameters 24 to 30 for motor 2
7	Any other TPU function	—
8	Any other TPU function	—
9	Any other TPU function	—
10	Any other TPU function	—
11	Any other TPU function	—
12	Any other TPU function	—
13	Motor 3, Master	Motor 3 control parameters
14	Motor 3, Slave 1	Acceleration parameters 1 to 16 for motor 3
15	Motor 3, Slave 2	Acceleration parameters 17 to 32 for motor 3

Table 5 Eight Motors
Each Controlled by Two Channels (Full-Step)
Local Table Configuration

Channel	Function	Parameter RAM Content
0	Motor 1, Master	Motor 1 control parameters
1	Motor 1, Slave	Acceleration parameters 1 to 12 for motor 1
2	Motor 2, Master	Motor 2 control parameters
3	Motor 2, Slave	Acceleration parameters 1 to 12 for motor 2
4	Motor 3, Master	Motor 3 control parameters
5	Motor 3, Slave	Acceleration parameters 1 to 12 for motor 3
6	Motor 4, Master	Motor 4 control parameters
7	Motor 4, Slave	Acceleration parameters 1 to 12 for motor 4
8	Motor 5, Master	Motor 5 control parameters
9	Motor 5, Slave	Acceleration parameters 1 to 12 for motor 5
10	Motor 6, Master	Motor 6 control parameters
11	Motor 6, Slave	Acceleration parameters 1 to 12 for motor 6
12	Motor 7, Master	Motor 7 control parameters
13	Motor 7, Slave	Acceleration parameters 1 to 12 for motor 7
14	Motor 8, Master	Motor 8 control parameters
15	Motor 8, Slave	Acceleration parameters 1 to 12 for motor 8

7 Table Stepper Motor Examples

Following are two example configurations of the TSM function, showing initial parameter RAM and control register contents and hardware connections to the motor. Many other configurations are possible.

7.1 Example A

7.1.1 Description

Configure the TSM function to drive a two-phase stepper motor in full-step mode using channels 3 and 4. Over the speed range that it will be operating, the motor requires 12 acceleration steps including the start rate and slew rate. The start rate has a step period equivalent to $4500 TCR1 clocks and the slew rate of the motor has a slew period equivalent to $1000 TCR1 clocks. The initial position of the motor should be set to $8000 with channel 3 outputting a high level and channel 4 a low level. The motor will be supplied with new desired positions via a CPU interrupt routine.

7.1.2 Initialization

Disable channels 3 and 4 by clearing priority bits. Select TSM function by programming the TSM function number into the function select registers. Enable interrupts on channels 3 and 4. Load the parameter RAM of channel 3 as shown below. Since the motor only requires ten acceleration parameters, local table configuration can be used. Load the parameter RAM of channel 4 with an acceleration table of choice as shown. Write HSQ = %00 to channel 3 to select local table configuration and one rotate of PIN_SEQUENCE between master and slave. Issue HSR = %10 to channel 3 and HSR = %01 to channel 4 to initialize the function and set up the initial pin levels. Write the priority bits of both channels to the same non-zero value.

Table 6 Channel 3 (Master) Parameter RAM

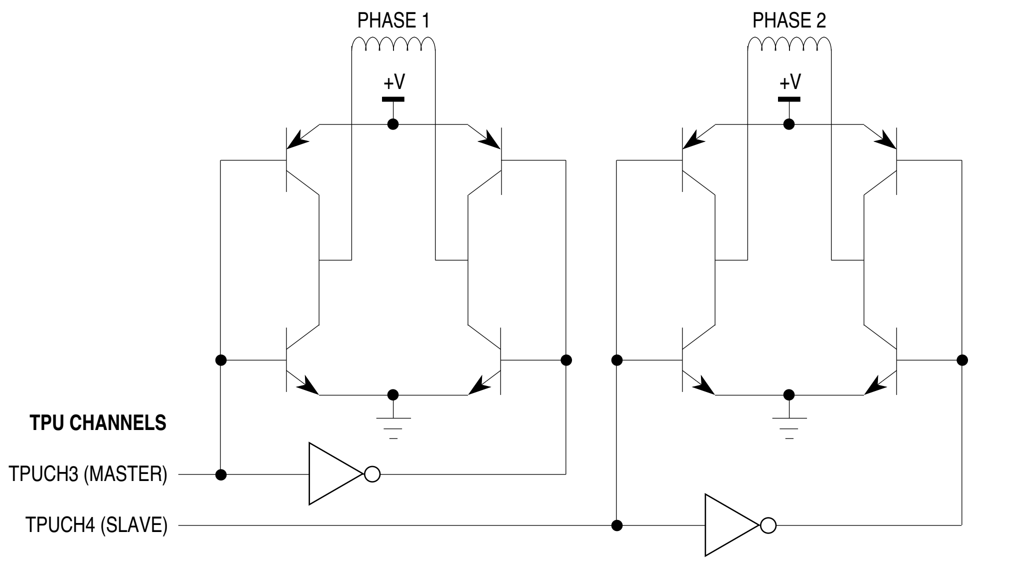
	15							8								0	
$YFFF30	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DESIRED_POSITION
$YFFF32	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CURRENT_POSITION
$YFFF34	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	
$YFFF36	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
$YFFF38	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	
$YFFF3A	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	

DESIRED_POSITION = CURRENT_POSITION = $8000, TABLE_SIZE = 10, TABLE_INDEX = 0, SLEW_PERIOD = $1000, BIT_S = 0, START_PERIOD = $4500, BIT_A = 0, PIN_SEQUENCE = $CCCC

Table 7 Channel 4 (Slave) Parameter RAM

	15	8 0	
$YFFF40	ACCEL_RATIO_2	ACCEL_RATIO_1	Acceleration
$YFFF42	ACCEL_RATIO_4	ACCEL_RATIO_3	Table
$YFFF44	ACCEL_RATIO_6	ACCEL_RATIO_5	
$YFFF46	ACCEL_RATIO_8	ACCEL_RATIO_7	
$YFFF48	ACCEL_RATIO_10	ACCEL_RATIO_9	
$YFFF4A	x	x	

7.2 Typical Hardware Configuration



7.3 Driving the Motor

The TPU will generate an interrupt request from each channel after it is initialized. When both interrupt requests have been received, the CPU can assume that the motor is correctly initialized. The CPU should then disable interrupts on the slave channel before issuing the first move request. To issue a move request the CPU should write DESIRED_POSITION and then issue an HSR %11 to channel 3. The CPU can repeat this process at any time. When the TSM function has moved the motor to the latest DESIRED_POSITION an interrupt request from channel 3 will be generated.

7.4 Example B

Configure the TSM function to drive two identical two-phase stepper motors in half-step bipolar mode using eight channels. Over the operating speed range, motor A requires 30 acceleration steps, including the start rate and slew rate, and motor B requires 70 acceleration steps. The start rate and slew rates of motor A have periods equivalent to $6800 and $2000 TCR1 clocks respectively. The start rate and slew rates of motor B have periods equivalent to $6800 and $500 TCR1 clocks respectively. The initial position of both motors should be set to $0000. Motor A is driven on a polling basis by the CPU and motor B is interrupt driven.

Since both motors require > 12 acceleration steps (> 14 including start and slew), split table configuration is used, and since both motors can share all the parameters, the most efficient TPU channel usage is obtained by assigning one motor to channels 1, 2, 3 and 4 (motor A) and the other to channels 13, 14, 15 and 0 (motor B). Disable all of the above channels by clearing their priority bits. Select TSM function by programming the TSM function number into the function select registers. Enable interrupts on channels 13, 14, 15 and 0. Load the parameter RAM of channels 1 and 13 as shown below. Load the parameter RAM of channels 2, 3, 4, 14, and 15 with an acceleration table of choice as shown (since only 68 parameters are required, channel 0 parameter RAM is not used). Write HSQ = %11 to channels 1 and 13 to select split table configuration and two rotates of PIN_SEQUENCE between channels. Both motors are being initialized with PIN_SEQUENCE = $E0E0, so issue an HSR = %10 to channels 1, 4, 13, and 0 and an HSR = %01 to channels 2, 3, 14 and 15 to initialize the function and set up the initial pin levels. Write the priority bits of the channels associated with each motor to the same non-zero value.

Table 8 Channel 1 (Motor A Master) Parameter RAM

	15							8								0	
$YFFF10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DESIRED_POSITION
$YFFF12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CURRENT_POSITION
$YFFF14	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	
$YFFF16	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
$YFFF18	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	
$YFFF1A	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	

DESIRED_POSITION = CURRENT_POSITION = $0000, TABLE_SIZE = 28, TABLE_INDEX = 0, SLEW_PERIOD = $2000, BIT_S = 0, START_PERIOD = $6800, BIT_A = 1, PIN_SEQUENCE = $E0E0

Table 9 Channel 13 (Motor B Master) Parameter RAM

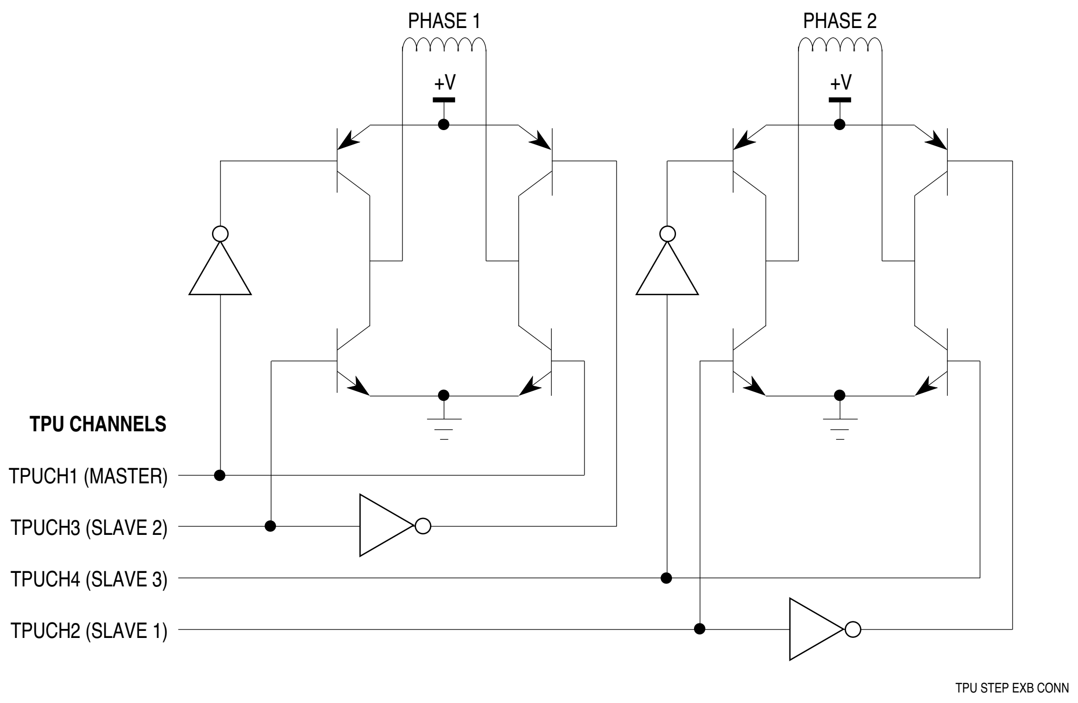
	15							8								0	
$YFFF10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DESIRED_POSITION
$YFFF12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CURRENT_POSITION
$YFFF14	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
$YFFF16	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	
$YFFF18	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	
$YFFF1A	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	

DESIRED_POSITION = CURRENT_POSITION = $0000, TABLE_SIZE = 68, TABLE_INDEX = 0, SLEW_PERIOD = $500, BIT_S = 0, START_PERIOD = $6800, BIT_A = 1, PIN_SEQUENCE = $E0E0

Table 10 Slave Parameter RAM — Acceleration Table

	15	8 0	
$YFFF20	ACCEL_RATIO_2	ACCEL_RATIO_1	Channel 2
$YFFF22	ACCEL_RATIO_4	ACCEL_RATIO_3	
$YFFF24	ACCEL_RATIO_6	ACCEL_RATIO_5	
$YFFF26	ACCEL_RATIO_8	ACCEL_RATIO_7	
$YFFF28	ACCEL_RATIO_10	ACCEL_RATIO_9	
$YFFF2A	ACCEL_RATIO_12	ACCEL_RATIO_11	
$YFFF30	ACCEL_RATIO_14	ACCEL_RATIO_13	Channel 3
$YFFF32	ACCEL_RATIO_16	ACCEL_RATIO_15	
$YFFF34	ACCEL_RATIO_18	ACCEL_RATIO_17	
$YFFF36	ACCEL_RATIO_20	ACCEL_RATIO_19	
$YFFF38	ACCEL_RATIO_22	ACCEL_RATIO_21	
$YFFF3A	ACCEL_RATIO_24	ACCEL_RATIO_23	
$YFFF40	ACCEL_RATIO_26	ACCEL_RATIO_25	Channel 4
$YFFF42	ACCEL_RATIO_28	ACCEL_RATIO_27	
$YFFF44	ACCEL_RATIO_30	ACCEL_RATIO_29	
$YFFF46	ACCEL_RATIO_32	ACCEL_RATIO_31	
$YFFF48	ACCEL_RATIO_34	ACCEL_RATIO_33	
$YFFF4A	ACCEL_RATIO_36	ACCEL_RATIO_35	
$YFFFE0	ACCEL_RATIO_38	ACCEL_RATIO_37	Channel 14
$YFFFE2	ACCEL_RATIO_40	ACCEL_RATIO_39	
$YFFFE4	ACCEL_RATIO_42	ACCEL_RATIO_41	
$YFFFE6	ACCEL_RATIO_44	ACCEL_RATIO_43	
$YFFFE8	ACCEL_RATIO_46	ACCEL_RATIO_45	
$YFFFEA	ACCEL_RATIO_48	ACCEL_RATIO_47	
$YFFFEC	ACCEL_RATIO_50	ACCEL_RATIO_49	
$YFFFEE	ACCEL_RATIO_52	ACCEL_RATIO_51	
$YFFFF0	ACCEL_RATIO_54	ACCEL_RATIO_53	Channel 15
$YFFFF2	ACCEL_RATIO_56	ACCEL_RATIO_55	
$YFFFF4	ACCEL_RATIO_58	ACCEL_RATIO_57	
$YFFFF6	ACCEL_RATIO_60	ACCEL_RATIO_59	
$YFFFF8	ACCEL_RATIO_62	ACCEL_RATIO_61	
$YFFFFA	ACCEL_RATIO_64	ACCEL_RATIO_63	
$YFFFFC	ACCEL_RATIO_66	ACCEL_RATIO_65	
$YFFFFE	ACCEL_RATIO_68	ACCEL_RATIO_67	

7.5 Typical Motor A Hardware Configuration



7.6 Driving the Motor

The TPU generates an interrupt request from each channel and clears the appropriate host service request bits after each channel is initialized. When all host service bits of motor A channels and all interrupt requests have been received from the motor B channels, the CPU can assume that both motors are correctly initialized (interrupt requests are generated by both motors, but only motor B has interrupts enabled).

To issue a move request to either motor, the CPU should write the relevant DESIRED_POSITION and then issue an HSR %11 to channel 1 for motor A or channel 13 for motor B. The CPU can repeat this process at any time. When the TSM function has moved motor A to DESIRED_POSITION, CURRENT_POSITION equals DESIRED_POSITION. When the TSM function has moved motor B to DESIRED_POSITION, CURRENT_POSITION equals DESIRED_POSITION and an interrupt request from channel 13 is generated.

In a real application, the choice of whether to control a motor by polling or interrupts depends on the importance of the motor in the system and the speed of response required. Where high performance control is required, interrupt driven control would be the normal choice.

8 TSM Function 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 regarding downloading and compiling microcode.

The table stepper motor function consists of four states, which operate as described below. For clarity, reference is made to internal flags (stepping and direction) in the following descriptions. These internal TPU control bits are not available to the user.

8.1 STATE1 —TSM_INIT_LO

This state, entered as a result of an HSR %01, configures the channel as an output low.

```
The channel pin is configured as an output
The pin is forced low
Channel flag 0 (stepping flag) is negated
All match and transition service requests are disabled
An interrupt request is generated
The state ends
```

8.2 STATE2 —TSM_INIT_HI

This state, entered as a result of an HSR %10, configures the channel as an output high.

```
The channel pin is configured as an output
The pin is forced high
Channel flag 0 (stepping flag) is negated
All match and transition service requests are disabled
An interrupt request is generated
The state ends
```

8.3 STATE3 —TSM_MOV_REQ

This state, entered as a result of an HSR %11, initiates a move if the motor is not already stepping.

```
MOVE_REQ
    Enable match and transition service requests
    If stepping_flag = 1 then do nothing and exit          (* skip dir_flag setup? *)
    Set stepping_flag                                      (* remember we are now stepping *)
    ERT = TCR1                                             (* load reference for first step match *)

SETUP_DIR
    temp = DESIRED_POSITION – CURRENT_POSITION
    If temp[msb] = 1 then
        clear dir_flag
        goto MOVE_DIRB                                     (* which way? *)
    Else
        If temp - 0 then
            set dir_flag                                   (* remember direction *)
            goto MOVE_DIRA
        Endif
        Endif
    Request interrupt                                      (* already there, no need to move *)
    clear stepping_flag
    Exit
```

8.4 STATE4 —TSM_STEP_MATCH

This state, entered as a result of a match on the master channel, evaluates the current position of the motor and either generates no step if the move is complete or generates a step with a period derived from an acceleration/slew/deceleration profile.

```
STEP
    If dir_flag = 1 then                                    (*step done: Update current position*)
        CURRENT_POSITION = CURRENT_POSITION + 1
    Else
        CURRENT_POSITION = CURRENT_POSITION – 1
    Endif
    If TABLE_INDEX = 0 then goto SETUP_DIR
                        (* Set up direction if finished decelerating and check for end of move*)
TST_DIR
    temp = DESIRED_POSITION – CURRENT_POSITION
    If temp[msb] = 1 then goto MOVE_DIRB
    Else                                                    (* which way should we be going? *)
        If temp - 0 then goto MOVE_DIRA
        Else
            If dir_flag = 0 then goto MOVE_DIRB
        Endif
    Endif          (* If DESIRED_POSITION = CURRENT_POSITION but TABLE_INDEX - 0 then
keep going in current direction*)

MOVE_DIRA
    If dir_flag = 0 then goto DECEL              (* If currently going in Direction B then decelerate*)
    If CURRENT_POSITION + TABLE_INDEX + 1 > DESIRED_POSITION then goto DECEL
    If CURRENT_POSITION + TABLE_INDEX + 1 = DESIRED_POSITION then goto NO_CHNGE

ACCEL
    If TABLE_INDEX ≤ TABLE_SIZE then                          (* more accel steps available? *)
            TABLE_INDEX = TABLE_INDEX + 1      (* accelerate by moving up table on each step*)
            goto CALC_PER
    Else
            TABLE_INDEX = TABLE_INDEX + 1
            goto TST_SLEW           (* If end of acceleration table reached then use slew period *)
    Endif

MOVE_DIRB
    If dir_flag = 1 then goto DECEL              (* If currently going in Direction A then decelerate*)
    If CURRENT_POSITION – TABLE_INDEX – 1 < DESIRED_POSITION then goto DECEL
    If CURRENT_POSITION – TABLE_INDEX – 1 = DESIRED_POSITION then goto NO_CHNGE
    goto ACCEL

NO_CHNGE
    If BIT_S = 0 then                                       (* use same step period as last time *)
        goto CALC_PER
    Else
        temp = SLEW_PERIOD                           (* If slewing then do not use lookup table *)
        goto SETUP_PINS
    Endif

TST_SLEW
    temp = SLEW_PERIOD
    If BIT_S = 0 then                                                   (* already slewing?*)
```

```
            set BIT_S                                      (* remember we are now *)
            goto SETUP_PINS
    Else
        TABLE_INDEX = TABLE_INDEX – 1                  (* do not go past TABLE_SIZE + 1*)
        goto SETUP_PINS
    Endif

DECEL
    TABLE_INDEX = TABLE_INDEX – 1          (* decelerate by moving back through table *)

CALC_PER
    If TABLE_INDEX = 0 or 1 then
        temp = START_PERIOD                (*Use start period or calculate step period? *)
        goto CLR_SLEW
    Endif
    temp = TABLE_INDEX – 1         (* – calculate, so this is correct table index for look up*)
```

(* Look-up obtains the acceleration parameter from the table. If local table configuration is selected then all parameters are obtained from contiguous RAM starting in the first slave channel. In split table configuration, the table is split between slave channel parameter RAM and the parameter RAM of channels 14, 15, 0. In two-channel mode the first 12 parameters are obtained from slave 1 and the rest from channels 14 to 0. In four-channel mode, the first 12 are obtained from slave 1, the next 12 from slave 2, the next 12 from slave 3 and the rest from channels 14 to 0. In split table configuration if channel 13 is the master then the slave resident parameters are obtained from channel 2 [and 3 and 4 in four-channel mode]. This is to allow channels 14 to 0 be used for the shared table parameters. *)

```
LOOK_UP
    If HSQ0 = 0 then                      (* where is accel parameter? in local or split table?*)
        get ACCEL_RATIO via temp from PRAM starting in slave1
        goto MUL_PER                                                          (* Local *)
    Endif
                                                                  (* split local and remote *)
    If BIT_A = 0 and temp > 11 then                               (* BIT_A: 2 or 4 channels *)
        get ACCEL_RATIO via temp from chan14⇒chan0 PRAM
        goto MUL_PER
    Endif
    If BIT_A = 1 and temp > 35 then
        get ACCEL_RATIO via temp from chan14⇒chan0 PRAM
        goto MUL_PER
    Endif
    If temp ≤ 11 then
        If chan_reg = 13 then                       (* master channel = 13 is a special case *)
            get ACCEL_RATIO via temp from channel 2 PRAM
        Else
            get ACCEL_RATIO via temp from slave1 PRAM
        Endif
        goto MUL_PER
    Endif
    If temp ≤ 23 then
        If chan_reg = 13 then
            get ACCEL_RATIO via temp from channel 3 PRAM
        Else
            get ACCEL_RATIO via temp from slave2 PRAM
        Endif
        goto MUL_PER
    Endif
    If chan_reg = 13 then
        get ACCEL_RATIO via temp from channel 4 PRAM
    Else
        get ACCEL_RATIO via temp from slave3 PRAM
    Endif

MUL_PER
    temp = (ACCEL_RATIO ∗ START_PERIOD)/256
                                  (* fractional multiply to calculate step period *)
    If temp ≤ SLEW_PERIOD then goto TST_SLEW            (* reached slew rate yet? *)

CLR_SLEW
    Clear BIT_S

SETUP_PIN S
    temp = temp + ERT
    temp1 = PIN_SEQUENCE
    temp2 = number of pins                                             (* 2 or 4 *)
    If dir_flag = 1 then
                   (* rotate PIN_SEQUENCE left or right depending on motor direction *)
        temp1 = R > temp1
    Else
        temp1 = R < temp1
    Endif
    PIN_SEQUENCE = temp1                                (* store new PIN_SEQUENCE *)

PIN_LOOP
    If temp1[msb] = 1 then
        configure pin to go high on next match
    Else
        configure pin to go low on next match
    Endif
    Set match for time temp                              (* set up match for next step *)
    If HSQ1 = 0 then
                          (* one or two rotates of PIN_SEQUENCE between slaves? *)
        temp1 = R > temp1
    Else
        temp1 = R >> temp1
    Endif
    channel = channel + 1                      (* move TPU operation to next channel *)
    temp2 = temp2 – 1
    If temp2 > 0 then goto PIN_LOOP                 (* more channels still to set up? *)
    Exit                                         (* DONE — next step is scheduled *)
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor