Freescale Semiconductor

Application Note

AN2364/D
Rev. 0, 10/2002

Using the Table Stepper Motor TPU Function (TSM) with the MPC500 Family

Glann Jackson
TECD Applications

This TPU Programming Note is intended to provide simple C interface routines to the table stepper motor TPU function (TSM). The routines are targeted for the MPC500 family of devices, but they should be easy to use with any device that has a TPU.

1 Functional Overview

The Table Stepper Motor (TSM) function is just one of numerous useful functions with pre-defined parameters available for the developer of TPU applications.These functions are located in the TPU ROM and are designated by a unique function number. The function number for the TSM is 0xD. Details of this appnote will instruct the user on calling the functions of the TSM and configuring these functions as needed.

The TSM function provides motor control to set the motor to a desired position (or step) in the 360 degree arch of motion. A relative distance between the current position and the desired position of the motor will determine the direction and amount of acceleration that the motor will be engaged to reach the desired position in the shortest amount of time. The values of acceleration are stored into a table in memory. Hence the name Table Stepper Motor function.

The 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 the motor 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 configured by the user via a variable length table that offers up to 96 step rates. The TPU can control up to eight motors in full step mode or 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 will result, 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. e.g. if a motor is currently 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 in the counterclockwise direction to reach position C.

The TSM generates the actual step patterns to drive the motor via synchronized output matches on two or four channels. The step patterns generated are 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 the slave(s) are then defined one or three channels immediately after the master in numeric order. For example, in two channel mode, if channel 5 was 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 the slaves. The choice of two or four channel mode is made via a control bit on the master channel.

The TSM function uses the same user defined step period profile during acceleration and deceleration. The user specifies this profile via a table 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 the 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 is programmable by the user. Limits for the various modes are explained below. Figure 2 illustrates these points.

2.1 Local Table Mode

In local table configuration, the 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.

The maximum table size is 48 bytes. This is determined by the amount of contiguous parameter RAM available, starting with parameter 0 of the first slave channel. The maximum number of step rates is equal to the table size plus 2 (start/stop and slew).

Note that the length of table is independent of whether two or four channel mode is selected e.g. if channel 14 is the master in a two channel configuration, the PRAM 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). Channels 14 and 15 will be the two motor driving channels.

2.2 Split Table Mode

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 to figure1). Since all slave parameter RAM is used in this mode, the 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. The parameters for channels 14, 15, and 0 are shared by all motors. Since the 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 the parameter RAM of channels 14, 15 and 0 is used to form the upper part of the acceleration profile, a special case exists if channel 13 is chosen as a master. This is caused by the fact that the sequential channels to 13 are in use. This case is handled as described below.

When operating in four-channel mode, split table configuration cannot be used with channel 11 or 12 programmed as the master channel unless the table length is equal or less than 32 or 16 bytes respectively.

2.2.1 Two-Channel Mode

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

When channel 13 is not the master channel, acceleration parameters 1 to 16 are obtained from the parameter RAM of the first slave channel (immediately after the master channel), starting with acceleration parameter 1 in the lower byte of parameter word 0.

When channel 13 is the master channel, acceleration parameters 1 to 16 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 the table size exceeds 16 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 16 ‘local’ parameters is 64 bytes, giving 66 step rates including start/stop and slew.

2.2.2 Four-Channel Mode

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

When channel 13 is not the master channel, acceleration parameters 1 to 16 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 17 to 32 are obtained from the parameter RAM of the second slave (2 channels after the master), starting with acceleration parameter 17 in the lower byte of parameter word 0. Acceleration parameters 33 to 48 are obtained from the parameter RAM of the third slave (3 channels after the master), starting with acceleration parameter 33 in the lower byte of parameter word 0.

When channel 13 is the master channel, acceleration parameters 1 to 16 are obtained from the parameter RAM of channel 2, starting with acceleration parameter 1 in the lower byte of parameter word 0. Acceleration parameters 17 to 32 are obtained from the parameter RAM of channel 3, starting with acceleration parameter 17 in the lower byte of parameter word 0. Acceleration parameters 33 to 48 are obtained from the parameter RAM of channel 4, starting with acceleration parameter 33 in the lower byte of parameter word 0.

In both cases when the table size exceeds 48 parameters the remainder is obtained from the contiguous parameter RAM of channels 14, 15, and 0, starting in the lower byte of parameter 0 of channel 14. In this configuration the maximum table length including the 48 ‘local’ parameters is 96 bytes, giving 98 step rates including start/stop and slew.

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:

$$97\ \mu\ \text{instructions} + 8\ \text{entries} = 105\ \text{long words}$$

4 TSM Function Structure

4.1 TSM Function Parameters Address Maps

This section provides detailed descriptions of TSM function parameters stored in channel parameter RAM. Table 1 shows TPU parameter RAM address mapping. 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).

Table 1. TPU Parameter RAM Address Mapping

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

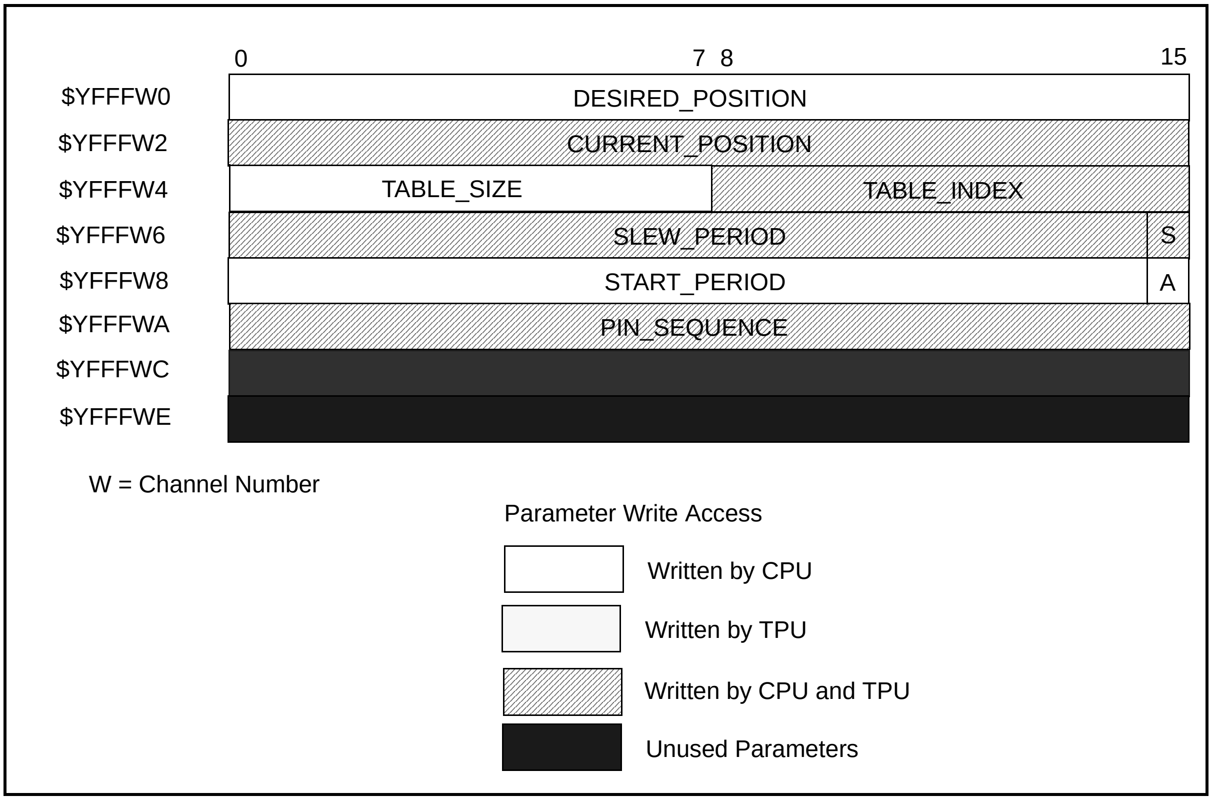


Figure 1. Master Channel Parameter Assignment—All Modes

Figure 1 shows the parameter RAM assignment used by the TSM function for the various modes of operation.

	0 7	8 15
$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+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
$YFFF(W+3)C	ACCEL_RATIO_46	ACCEL_RATIO_45
$YFFF(W+3)E	ACCEL_RATIO_48	ACCEL_RATIO_47

W = Master Channel Number

Figure 2. Acceleration Parameter Table—Local Configuration.

Figure 2 shows the positions in memory held by the acceleration parameters for a local configuration. The only parameter RAM locations used here are the slave channel parameter RAMs. Figure 3 shows the acceleration parameter address locations for a split table configuration in a two channel mode. The single slave parameter channel and channels 14, 15, and 0 are shown in this table.

		0 … 7	8 … 15
	$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+1)C	ACCEL_RATIO_14	ACCEL_RATIO_13
	$YFFF(W+1)E	ACCEL_RATIO_16	ACCEL_RATIO_15
CHANNEL 14	$YFFFE0	ACCEL_RATIO_18	ACCEL_RATIO_17
	$YFFFE2	ACCEL_RATIO_20	ACCEL_RATIO_19
	$YFFFE4	ACCEL_RATIO_22	ACCEL_RATIO_21
	$YFFFE6	ACCEL_RATIO_24	ACCEL_RATIO_23
	$YFFFE8	ACCEL_RATIO_26	ACCEL_RATIO_25
	$YFFFEA	ACCEL_RATIO_28	ACCEL_RATIO_27
	$YFFFEC	ACCEL_RATIO_30	ACCEL_RATIO_29
	$YFFFEE	ACCEL_RATIO_32	ACCEL_RATIO_31
CHANNEL 15	$YFFFF0	ACCEL_RATIO_34	ACCEL_RATIO_33
	$YFFFF2	ACCEL_RATIO_36	ACCEL_RATIO_35
	$YFFFF4	ACCEL_RATIO_38	ACCEL_RATIO_37
	$YFFFF6	ACCEL_RATIO_40	ACCEL_RATIO_39
	$YFFFF8	ACCEL_RATIO_42	ACCEL_RATIO_41
	$YFFFFA	ACCEL_RATIO_44	ACCEL_RATIO_43
	$YFFFFC	ACCEL_RATIO_46	ACCEL_RATIO_45
	$YFFFFE	ACCEL_RATIO_48	ACCEL_RATIO_47
CHANNEL 0	$YFFF00	ACCEL_RATIO_50	ACCEL_RATIO_49
	$YFFF02	ACCEL_RATIO_52	ACCEL_RATIO_51
	$YFFF04	ACCEL_RATIO_54	ACCEL_RATIO_53
	$YFFF06	ACCEL_RATIO_56	ACCEL_RATIO_55
	$YFFF08	ACCEL_RATIO_58	ACCEL_RATIO_57
	$YFFF0A	ACCEL_RATIO_60	ACCEL_RATIO_59
	$YFFF0C	ACCEL_RATIO_62	ACCEL_RATIO_61
	$YFFF0E	ACCEL_RATIO_64	ACCEL_RATIO_63

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

Figure 3. Acceleration Parameter Table—Split Table Configuration (2-Channel Mode)

Figure 4 shows the acceleration parameter address locations for the split table configuration in the four channel mode. This version will take the three slave parameter channels and channels 14, 15, and 0 for a total of six parameter channels and 96 bytes and a total of 98 step rates when including start/stop and slew.

		0 7	8 15
	$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+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+2)8	ACCEL_RATIO_42	ACCEL_RATIO_41
	$YFFF(W+3)A	ACCEL_RATIO_44	ACCEL_RATIO_43
	$YFFF(W+3)C	ACCEL_RATIO_46	ACCEL_RATIO_45
	$YFFF(W+3)E	ACCEL_RATIO_48	ACCEL_RATIO_47
CHANNEL 14	$YFFFE0	ACCEL_RATIO_50	ACCEL_RATIO_49
	$YFFFE2	ACCEL_RATIO_52	ACCEL_RATIO_51
	$YFFFE4	ACCEL_RATIO_54	ACCEL_RATIO_53
	$YFFFE6	ACCEL_RATIO_56	ACCEL_RATIO_55
	$YFFFE8	ACCEL_RATIO_58	ACCEL_RATIO_57
	$YFFFEA	ACCEL_RATIO_60	ACCEL_RATIO_59
	$YFFFEC	ACCEL_RATIO_62	ACCEL_RATIO_61
	$YFFFEE	ACCEL_RATIO_64	ACCEL_RATIO_63
CHANNEL 15	$YFFFF0	ACCEL_RATIO_66	ACCEL_RATIO_65
	$YFFFF2	ACCEL_RATIO_68	ACCEL_RATIO_67
	$YFFFF4	ACCEL_RATIO_70	ACCEL_RATIO_69
	$YFFFF6	ACCEL_RATIO_72	ACCEL_RATIO_71
	$YFFFF8	ACCEL_RATIO_74	ACCEL_RATIO_73
	$YFFFFA	ACCEL_RATIO_76	ACCEL_RATIO_75
	$YFFFFC	ACCEL_RATIO_78	ACCEL_RATIO_77
	$YFFFFE	ACCEL_RATIO_80	ACCEL_RATIO_79
CHANNEL 0	$YFFF00	ACCEL_RATIO_82	ACCEL_RATIO_81
	$YFFF02	ACCEL_RATIO_84	ACCEL_RATIO_83
	$YFFF04	ACCEL_RATIO_86	ACCEL_RATIO_85
	$YFFF06	ACCEL_RATIO_88	ACCEL_RATIO_87
	$YFFF08	ACCEL_RATIO_90	ACCEL_RATIO_89
	$YFFF0A	ACCEL_RATIO_92	ACCEL_RATIO_91
	$YFFF0C	ACCEL_RATIO_94	ACCEL_RATIO_93
	$YFFF0E	ACCEL_RATIO_96	ACCEL_RATIO_95

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

Figure 4. Acceleration Parameter Table—Split Table Configuration (4-Channel Mode)

4.1.1 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.

4.1.2 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 = 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.

4.1.3 TABLE_SIZE

This 8-bit parameter, initialized by the CPU, defines the length of the acceleration table. The valid range for TABLE_SIZE is 1 to maximum. In local table configuration, maximum is 16 in two channel mode. In local table configuration and four channel mode, the maximum is 48. In split table configuration, the maximum is 64 in two channel mode and 96 in four channel mode. Note that maximum will be reduced if a consecutive channel, that would have been a slave parameter table channel, is programmed as another TSM master channel or to run another TPU function. TABLE_SIZE should not be written while the motor is moving.

4.1.4 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 should 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.

4.1.5 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 should be written to zero by the CPU during initialization and then never written again.

4.1.6 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 will be determined by TPU latency and motor characteristics. SLEW_PERIOD should not be changed while the motor is moving.

4.1.7 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, 4 will be 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.

4.1.8 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 the 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. START_PERIOD should not be changed while the motor is moving.

4.1.9 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. This parameter 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 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 the 1 or 3 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 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.

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.

4.1.10 ACCEL_RATIO_1 ....ACCEL_RATIO _N

These 8-bit parameters make up the acceleration parameter table and determine the step periods during acceleration and deceleration phases. The CPU initializes these table parameters. The START_PERIOD parameter is fractionally multiplied by successive table values to determine the step periods during acceleration and deceleration. The step period obtained from ACCEL_RATIO_N is given by

STEP_PERIOD = (START_PERIOD * ACCEL_RATIO_N) / 256

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

4.1.11 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.

4.1.12 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, then one rotate is performed. If HSQ1 = 1, then two rotates are performed. HSQ1 should 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 TSM Function

This section provides information concerning the TPU host interface to the TSM function.Figure 5 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 = 0x7 or 0xF).

Address	0 7	8 15
$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 (HSSR0)	
$YFFE1A	HOST SERVICE REQUEST REGISTER 1 (HSSR1)	
$YFFE1C	CHANNEL PRIORITY CHANNEL 0 (CPR0)	
$YFFE1E	CHANNEL PRIORITY CHANNEL 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 5. TPU Address Map

CIER - Channel Interrupt Enable Register 0xYFFE0A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

Table 2. CIER Bit Settings

CH	Interrupt Enable
0	Channel interrupts disabled
1	Channel interrupts enabled

CFSR[0:3] - Channel Function Select Registers 0xYFFE0C–0xYFFE0E 0xYFFE10–0xYFFE12

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CFS(CH15, 11, 7, 3)				CFS(CH14, 10, 6 , 2)				CFS(CH13, 9, 5, 1)				CFS(CH12, 8, 4, 0)			

CFS[3:0] -- Function Number (Assigned during microcode assembly).

HSQR[0:1] - Host Sequence Registers 0xYFFE14--0xYFFE16

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CH15, 7		CH14, 6		CH13, 5		CH12, 4		CH11, 3		CH10, 2		CH9, 1		CH8, 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

When initializing the Host Sequence Register bits, a logical “OR” of the two bits per channel should be used in code. The initialization function in the TSM API interface performs this activity.

HSSR[0:1] - Host Service Request Registers 0xYFFE18–0xYFFE1A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CH15, 7		CH14, 6		CH13, 5		CH12, 4		CH11, 3		CH10, 2		CH9, 1		CH8, 0	

CH[15:0]	Initialization
00	No host service (Reset Condition)
01	Initialize, Pin low
10	Initialize, Pin high
11	Move Request (master only)

The Host Service Request Registers control the changes on the pins in states 01 and 10. The TSM function starts it activity of moving the current position to the desired position when state 11 is written to the two bits for the respective TSM master channel. After writing a state to the HSSR register, observing the value in the bits will show a quick return back to state 00.

CPR[0:1] - Channel Priority Registers 0xYFFE1C–0xYFFE1E

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CH15, 7		CH14, 6		CH13, 5		CH12, 4		CH11, 3		CH10, 2		CH9, 1		CH8, 0	

CH[15:0]	Channel Priority
00	Disabled
01	Low
10	Middle
11	High

The priority for the TSM channels should be coordinated with other functions in the TPU system.

CISR - Channel Interrupt Statue Register 0xYFFE20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

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

6 Configuration of TSM Function

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). These are the steps necessary to initialize and then run a TSM function with the channels of the TPU:

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 two or four channel operation. BIT_S should be cleared and never written too again.
8. PIN_SEQUENCE is written with a value that will determine the channel pin responses as the motor steps. Bit14 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 the 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)
Slave1	HSR %01	(pin low)
Slave2	HSR %01	(pin low)
Slave3	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)
Slave1	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.

7 TSM Routines

The following routines provide easy access, for the application developer, into the TSM function. Use of these functions eliminate the need to directly control the TPU registers. There are eight functions added to the application programming interface (API). The routines can be found in the tpu_tsm.h and tpu_tsm.c files which should be included in the link file along with the top level development file(s). The first four functions are used at the top application program level to directly operate the TSM function. The four other TSM functions provide utility sub-routine activities and data manipulation. The routines will be described in order and are listed below:

- void tpu_tsm_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, INT16 start_position, UINT16 table_size_index, UINT16 slew_period, UINT16 start_period, UINT16 pin_sequence, UINT8 number_channels, UINT16 *table, UINT8 table_size);
- void tpu_tsm_mov(struct TPU3_tag *tpu, UINT8 channel, UINT16 position);
- UINT16 tpu_tsm_rd_dp(struct TPU3_tag *tpu, UINT8 channel);
- UINT16 tpu_tsm_rd_cp(struct TPU3_tag *tpu, UINT8 channel);
- UINT16 tpu_tsm_mas_chan_cier(int master_chan);
- void tpu_tsm_int_lev(struct TPU3_tag *tpu, UINT8 level);
- int tpu_tsm_int_chk(struct TPU3_tag *tpu, UINT16 channel);
- void tpu_tsm_cisr_clr(struct TPU3_tag *tpu, UINT16 CISR_level);

void tpu_tsm_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, INT16 start_position, UINT16 table_size_index, UINT16 slew_period, UINT16 start_period, UINT16 pin_sequence, UINT8 number_channels, UINT16 *table, UINT8 table_size);

7.1 void tpu_tsm_init

This routine is used to initialize the channels of the TPU for the TSM function. This function has 11 input parameters. Do not let the large number scare one away from using this function. Each function will be handled one at a time and will handle all of the initialization requirements of the TSM function.

- *tpu -- This is the pointer to the TPU module chosen to run the TSM function. It is a structure of type (name) TPU3_tag which is defined in m_tpu3.h.
- channel -- This is the channel number of the primary TSM channel. The following channels will be initialized as the parameter table channels.
- priority -- This is the priority level which is assigned to all channels used for this TSM function. This parameter should be assigned a value of: TPU_PRIORITY_HIGH, TPU_PRIORITY_MIDDLE, or TPU_PRIORITY_LOW. The TPU priorities are defined in the file mpc500_utils.h.

- start_position -- This is a 16-bit integer which establishes the initial value for both the DESIRED_POSITION and the CURRENT_POSITION. This is efficient since both values need to be set to the same value when the TSM function is initialized.
- table_size_index -- This parameter combines the table size and the table index values into a single 16-bit input. The eight bits for the table size is the number of bytes used in the parameter table encoded into a hex ($) value. The table index is set in the last 8-bits of this parameter and must be set to the value of zero.
- slew_period -- This parameter combines the slew period with the 1-bit "S" value. The slew period value must be shifted left by one bit after encoding into a hex value. e.g. an original value of $2000 will be encoded as $4000.The least significant bit (bit #15, Big Endian) is the "S" bit and must always be written to zero (and only at initialization).
- start_period -- This parameter combines the start period with the 1-bit "A" value. The start period value must be shifted left by one bit after encoding into a hex value. e.g. an original value of $6800 will be encoded as $D000. The least significant bit (bit #15, big endian) is the "A" bit. A value of 0 will initialize a two channel TSM function and a value of 1 will initialize a four channel function. This value must not be changed after initialization.
- pin_sequence -- This parameter determines the step patterns that are output on two or four TPU pins. Two channel and four channel example values are $3333 and $E0E0 respectively.
- number_channels -- This parameter is used with the master channel designation to determine which channels will be the parameter table channels for the TSM function. Either a two or four channel designation is valid. The master channel is included in this number.
- *table -- This is the pointer to the first parameter table channel. This is used to load the parameter table for the TSM from memory into the TPU RAM.
- table_size -- This parameter provides a byte count limit when loading the parameter table.

The function does not return any values from initialization. Except for DESIRED_POSITION, the parameters of this initialization function must not be changed while the TSM function is operating. After initialization, the parameter table channels will never be rewritten. Only the master channel will be accessed for any future control commands of the TSM function.

7.2 void tpu_tsm_mov

This routine is the only control sent to the TSM function after initialization. Basically, this function will designate where to move the stepper motor. This is accomplished with the following input parameters:

- *tpu -- This is the pointer to the TPU module chosen to run the TSM function. It is a structure of type (name) TPU3_tag which is defined in m_tpu3.h.
- channel -- This is the channel number of the primary TSM master channel.
- position -- This parameter is the value of the new DESIRED_POSITION.

After the new DESIRED_POSITION is written to the internal register, a call is made which will move the TSM until there is a match between the CURRENT_POSITION and the DESIRED_POSITION. Also, the move function can be called at any time; even while the motor is stepping towards another value. The TSM function will automatically take care of slowing the motor, reversing the direction (if necessary), and accelerating towards the new DESIRED_POSITION as needed.

7.3 UINT16 tpu_tsm_rd_dp

This routine will read the value of the DESIRED_POSITION. This value is used for program control when compared against some other value.

- *tpu -- This is the pointer to the TPU module chosen to run the TSM function. It is a structure of type (name) TPU3_tag which is defined in m_tpu3.h.
- channel -- This is the channel number of the primary TSM master channel.

The value of the DESIRED_POSITION is returned as a UINT16 function.

7.4 UINT16 tpu_tsm_rd_cp

This routine will read the value of the CURRENT_POSITION. This value is used for program control when compared against some other value.

- *tpu -- This is the pointer to the TPU module chosen to run the TSM function. It is a structure of type (name) TPU3_tag which is defined in m_tpu3.h.
- channel -- This is the channel number of the primary TSM master channel.

The value of the CURRENT_POSITION is returned as a UINT16 function.

7.5 UINT16 tpu_tsm_mas_chan_cier

This routine will return the UINT16 value of the CIER or CISR register encoding from the integer input of the master channel. The basic function of this routine is to perform the integer to register hex value conversion.

- master_chan -- This is the channel number of the primary TSM master channel.

This routine only has one input since it basically performs a single utility task. This routine is associated with the TSM function since it relates specifically to the master channel definition of the TSM.

7.6 void tpu_tsm_int_lev

This routine converts the chosen integer interrupt level value and applies this value to specific internal level values in the ILBS and CIRL registers. The level is chosen for this particular TSM function initialization. This routine is optionally used if the application development uses an interrupt structure.

- *tpu -- This is the pointer to the TPU module chosen to run the TSM function. It is a structure of type (name) TPU3_tag which is defined in m_tpu3.h.
- level -- This is the integer level of range 0 to 31 which will be invoked when the DESIRED_POSITION matches the CURRENT_POSITION.

The value of ILBS and CIRL of the TPU TICR register is not returned but directly entered into the register value. The developer does not need to worry about these details at this point of the program. The developer only needs to choose an interrupt level that will coherently fit into the system application being developed.

7.7 INT tpu_tsm_int_chk

This routine will read the value of the active interrupt channel in the CISR register. This value is compared to the input channel to determine a match. A match confirms that the highest priority active interrupt is for the specific TSM function.

- *tpu -- This is the pointer to the TPU module chosen to run the TSM function. It is a structure of type (name) TPU3_tag which is defined in m_tpu3.h.
- channel -- This is the channel number of the primary TSM master channel. The 16-bit input value is encoded to match the CISR value (see: *tpu_tsm_mas_chan_cier*()).

The integer value of TPU_TSM_TRUE or TPU_TSM_FALSE is returned depending upon the result of the compare.

7.8 void tpu_tsm_cisr_clr

This routine will clear the CISR register. This will have the effect of clearing all active interrupts in the CISR register.

- *tpu -- This is the pointer to the TPU module chosen to run the TSM function. It is a structure of type (name) TPU3_tag which is defined in m_tpu3.h.
- •CISR_level -- This is the channel number which should match the assigned interrupt level for the TSM function. This currently serves no function. However, further development could clear a specific channel instead of all channels of the CISR register.

This routine is needed at the end of the initialization routine to cover interrupts which were activated during initialization. An interrupt handling routine could take care of active interrupts before cancellation. Even if interrupts are not used with the TPU, these interrupts should be cleared to reduce the danger of a spurious interrupt being activated.

8 Performance and Use of TSM Function

8.1 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 Table 3 below.

Table 3. Table Stepper Motor Function—State Timing

State Number and Name	Max. CPU Clock Cycles	RAM Accesses by TPU
S1 -- TSM_INIT_LO	6	
S2 -- TSM_INIT_HI	6	
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

8.2 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 6) 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. The 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 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 channel bit in the initial PIN_SEQUENCE. For example, if $3333 is written to PIN_SEQUENCE, then an HSR%01 should be issued to the master channel (pin low) and an HSR%10 to the slave channel (pin high).

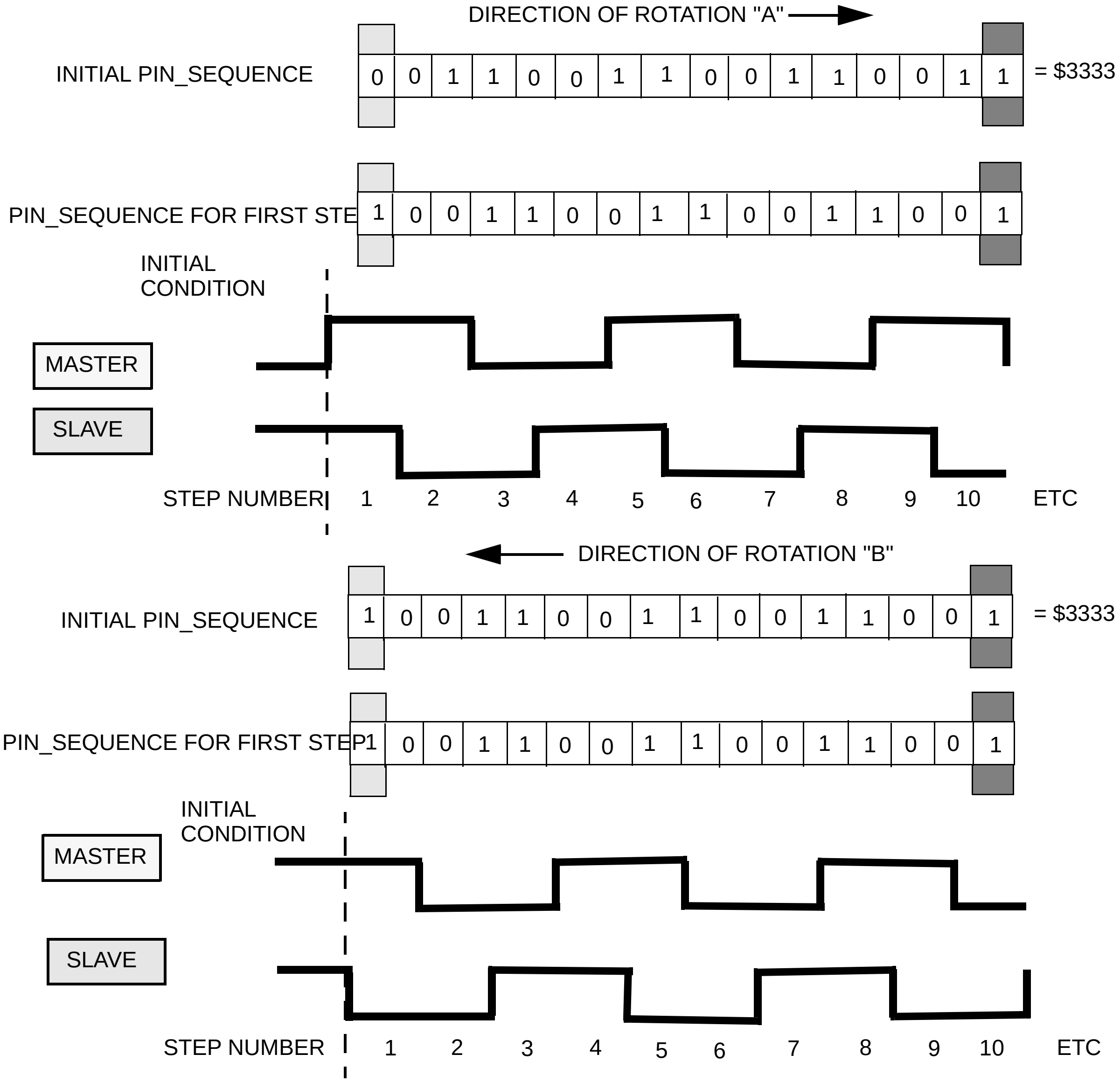


Figure 6. Two-Channel Mode—Full Step Generation

In half-step four-channel mode (see Figure 7), 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. The 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 7 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 channel 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)

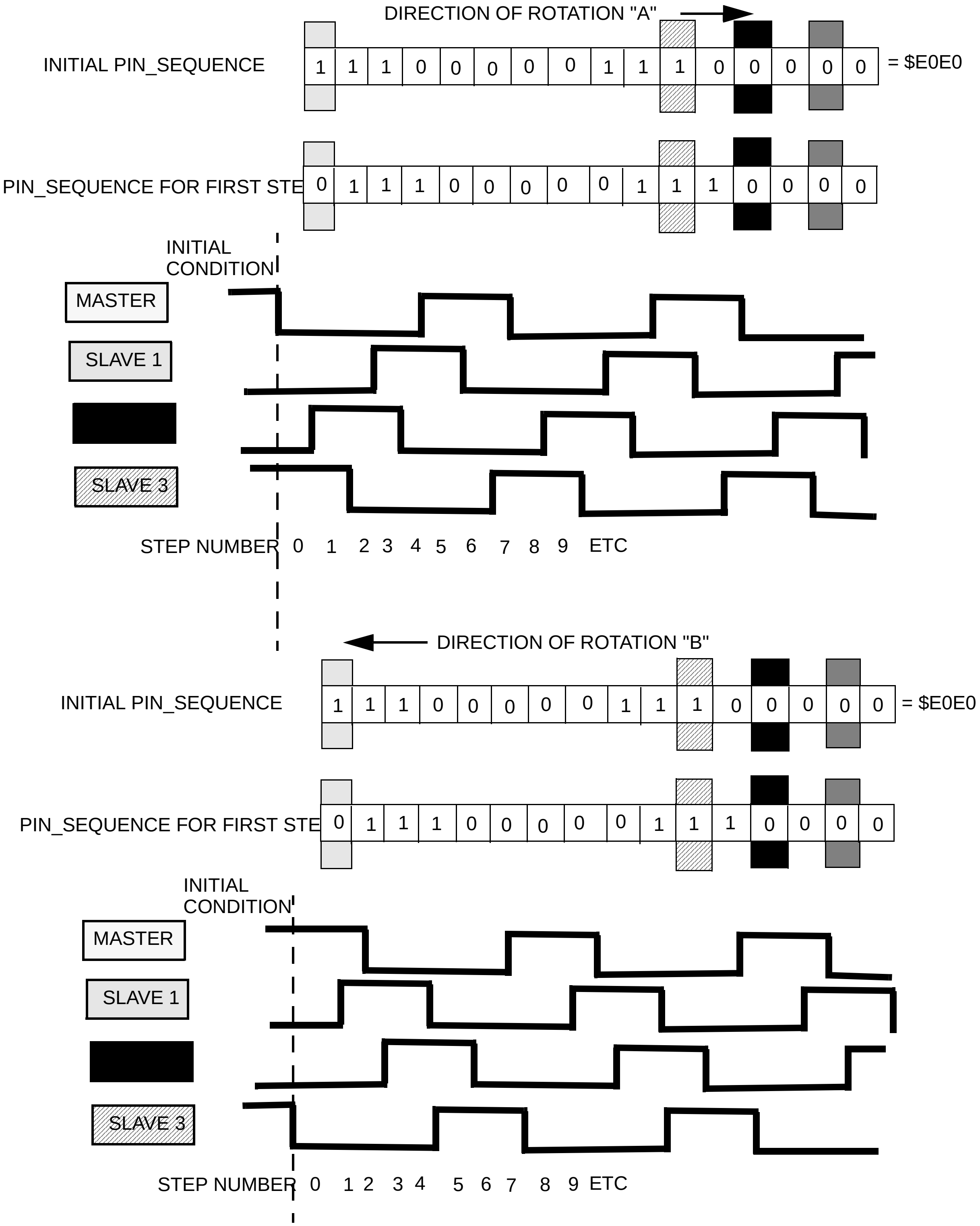


Figure 7. Four-Channel Mode—Half-Step Generation

9 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 the normal function and use of the TSM. More detail on the microcode operation is shown in the "C" coding examples at the end of this document.

9.1 Simple A -> 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 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)

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 form 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 will continue 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 8. 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.

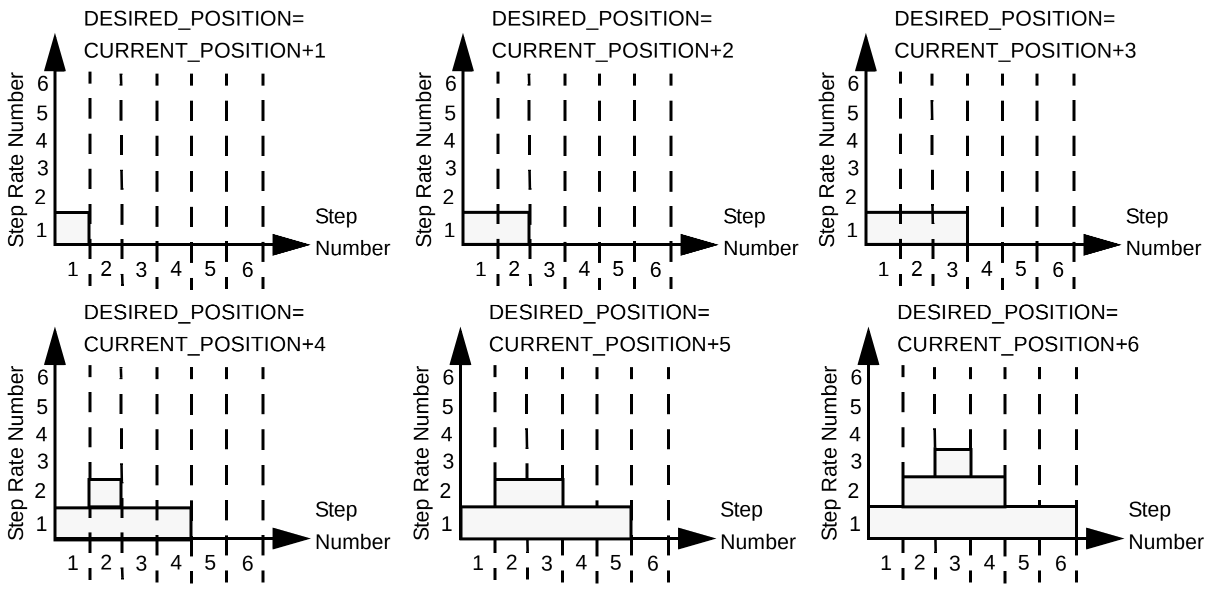


Figure 8. Short Move Position Algorithm Examples—No mid_move change

9.2 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 the same direction as old DESIRED_POSITION but further away. See Figure 9a.

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 the same direction as old DESIRED_POSITION but closer. See Figure 9b.

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, the motor will subsequently reverse direction and accelerate (if possible) to the new DESIRED_POSITION.

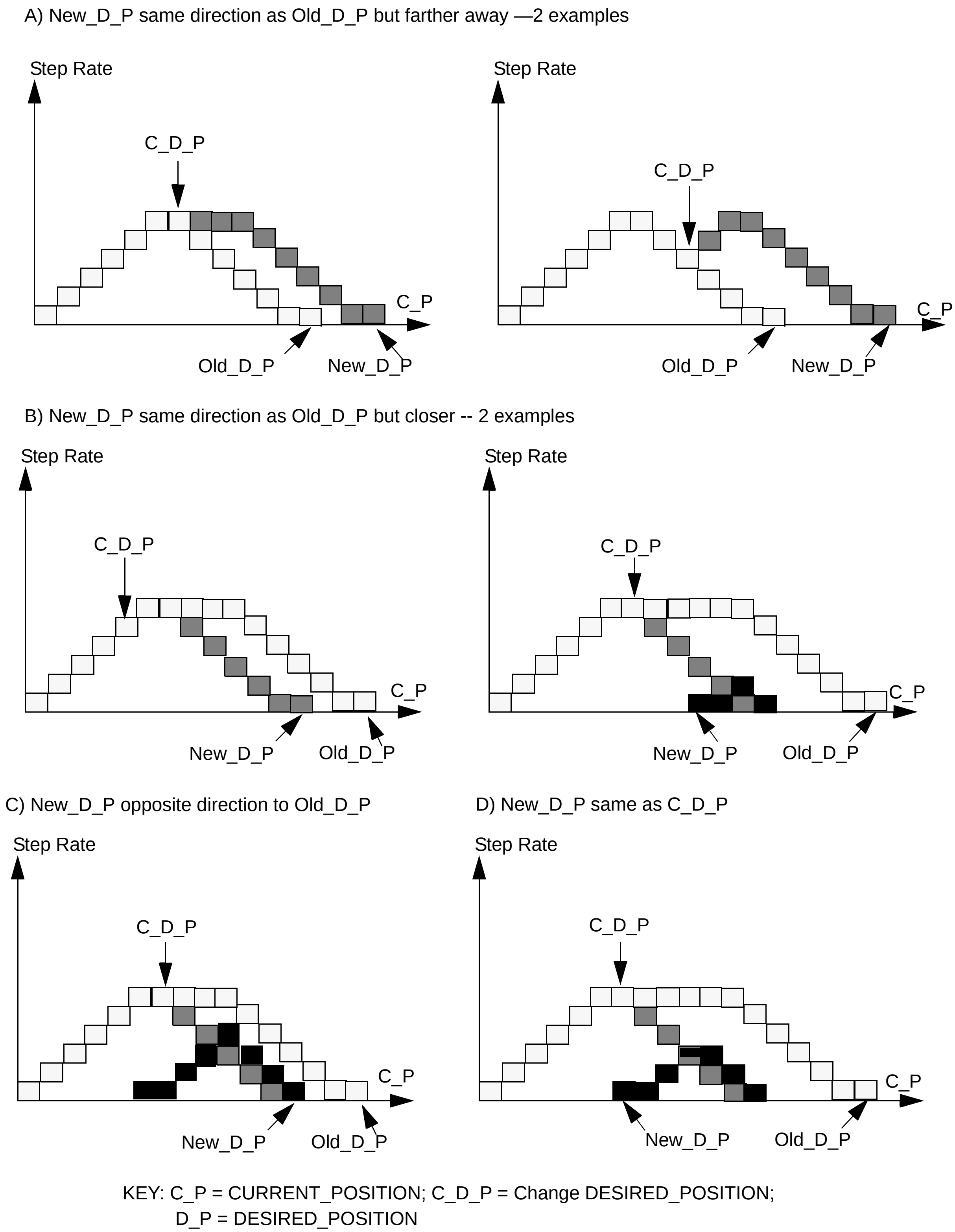
Case 3: New DESIRED_POSITION is in opposite direction to CURRENT_POSITION than old DESIRED_POSITION. See Figure 9c.

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 9d.

If the last step had a period equal to START_PERIOD, then 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) New_D_P same direction as Old_D_P but farther away —2 examples

B) New_D_P same direction as Old_D_P but closer -- 2 examples

C) New_D_P opposite direction to Old_D_P

D) New_D_P same as C_D_P

KEY: C_P = CURRENT_POSITION; C_D_P = Change DESIRED_POSITION;
D_P = DESIRED_POSITION

Figure 9. The Effect of Changing DESIRED_POSITION During Mid-move

9.3 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. The end of the acceleration table is reached.
2. 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.

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

9.4 Choosing Between the TSM Table Modes

The TPU has a limited amount 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 how many TPU channels are required for other functions. It is assumed that the application requires the maximum possible number of acceleration steps. If this is not the case, then a better arrangement may be possible.

10 Table Stepper Motor C Code Examples

There are three program listings that will be exhibited to show the TSM architecture programming interface (API). The two API programs are used with the Freescale MPC500 header files and are 1) tpu_tsm.c and 2) tpu_tsm.h. The third program is an example listing for the developer for their own application.

10.1 tpu_tsm.c

The tpu_tsm.c program follows:

```
/*=========================================================================*/
/* COMPILER: Diab Data        VERSION: 4.3g                                */
/* AUTHOR: Glenn Jackson                                                   */
/*                                                                         */
/* HISTORY                                                                 */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                   */
/* ---   -----------  ---------    ---------------------                   */
/* 1.0   G. Jackson   30/Jul/02    Initial version of function.            */
/***************************************************************************/
#include "tpu_tsm.h"
#include "mpc500_util.h"
```

```
/*************************************************************/
/*            Begin TPU Initialization                       */
/*************************************************************/

/*******************************************************************************
FUNCTION      : tpu_tsm_init
PURPOSE       : To initialize channels to run the TSM function.
INPUTS NOTES  : This function has 13 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                         type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the channel number of the primary TSM
                           channel. The next channels are used as the
                            parameter table.
                 priority - This is the priority to assign to TSM channels.
                            This parameter should be assigned a value of:
                            TPU_PRIORITY_HIGH, TPU_PRIORITY_MIDDLE or
                            TPU_PRIORITY_LOW.
                 start_position - This is the starting desired_position and current_position.
                 table_size - Initializes the number of bytes in the
                            acceleration table.
                 table_index - Initializes the table pointer to zero (start).
                 slew_period - initializes the slew_period and S.
                 start_period - initializes the start_period and A.
                 pin_sequence - initializes the pin_sequence.
                 number_channels - number of channels used for this TSM.
                            Two to four channels total, including the master
                            channel and the parameter table channels.
                 *table - pointer to the table beginning location.
                 table_size - initialize the number of bytes in the table.
RETURNS NOTES : none
WARNING       : The channels must be stopped before it is reconfigured. The
                function disables the channels but if they were currently
                being serviced it would continue. The delay for assigning the
                pram pointer may to enough but depends on system loading.
*******************************************************************************/
void tpu_tsm_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority,
                  INT16 start_position, UINT16 table_size_index,
                  UINT16 slew_period, UINT16 start_period,
                  UINT16 pin_sequence, UINT8 number_channels,
                  UINT16 *table, UINT8 table_size)
```

```
{

/* Declare variables for channel designations */
          int master_channel;
          int param1_channel;
          int param2_channel;
          int param3_channel;

    UINT8 pchannel;
    UINT8 param;
    UINT8 i;

    /* Establish channels used for TSM function */
    /* set channel values to other channels based upon the master */
   /* if primary channel is 15 then secondary channel should be 0 */
    master_channel = channel;
    param1_channel = (channel + 1) & 0xF;
    param2_channel = (channel + 2) & 0xF;
    param3_channel = (channel + 3) & 0xF;

    /* disable channels so they can be configured safely */
    tpu_disable( tpu, master_channel);
    tpu_disable( tpu, param1_channel);
    if (number_channels > 2) { tpu_disable( tpu, param2_channel); }
    if (number_channels > 3) { tpu_disable( tpu, param3_channel); }

    /* select TSM function for number of active channels */
    /*  TSM is function 0xD                  */
    tpu_func( tpu, master_channel, TPU_FUNCTION_TSM);
    tpu_func( tpu, param1_channel, TPU_FUNCTION_TSM);
    if (number_channels > 2) { tpu_func( tpu, param2_channel, TPU_FUNCTION_TSM); }
    if (number_channels > 3) { tpu_func( tpu, param3_channel, TPU_FUNCTION_TSM); }

    /* disable interupts on channels so they can be configured safely */
    tpu_interrupt_disable( tpu, master_channel);
    tpu_interrupt_disable( tpu, param1_channel);
    if (number_channels > 2) { tpu_interrupt_disable( tpu, param2_channel); }
    if (number_channels > 3) { tpu_interrupt_disable( tpu, param3_channel); }


    /* Initialize Parameter RAM   */
    pchannel = param1_channel;
    param = 0;

    for (i=0; i<table_size; i++){
        tpu->PARM.R[pchannel][param++] = *table++;
        if (param == 8){
            param = 0;
            pchannel++;
            if (pchannel == 16) pchannel = 0;
            }
        }


    /* -initial DESIRED_POSITION and CURRENT_POSITION will be the same;      */
    /* -initialize TPU_TSM_TABLE, slew period, start period, pin_sequence    */
    tpu->PARM.R[master_channel][TPU_TSM_DESIRED_POSITION] = (INT16) (start_position);
    tpu->PARM.R[master_channel][TPU_TSM_CURRENT_POSITION] = (INT16) (start_position);
    tpu->PARM.R[master_channel][TPU_TSM_TABLE]  = (INT16) (table_size_index);
    tpu->PARM.R[master_channel][TPU_TSM_SLEW_PERIOD] = (INT16) (slew_period);
    tpu->PARM.R[master_channel][TPU_TSM_START_PERIOD] = (INT16) (start_period);
    tpu->PARM.R[master_channel][TPU_TSM_PIN_SEQUENCE] = (INT16) (pin_sequence);

/**************************************************/
/*    Configure the Channels.                     */
/**************************************************/

    /* Configure the first channel as the primary channel and the following */
    /* channel as the secondary channel.                                    */
    tpu_hsq(tpu, master_channel, TPU_TSM_LOCAL_ACC_TBL | TPU_TSM_ROTATE_ONCE);
    tpu_hsq(tpu, param1_channel, TPU_TSM_LOCAL_ACC_TBL | TPU_TSM_ROTATE_ONCE);
    if(number_channels > 2) {
    tpu_hsq(tpu, param2_channel, TPU_TSM_LOCAL_ACC_TBL | TPU_TSM_ROTATE_ONCE);
    }
    if(number_channels > 3) {
    tpu_hsq(tpu, param3_channel, TPU_TSM_LOCAL_ACC_TBL | TPU_TSM_ROTATE_ONCE);
    }

    /* Initialize both channels */
    tpu_hsr(tpu, master_channel, TPU_TSM_INIT_HI);
    tpu_hsr(tpu, param1_channel, TPU_TSM_INIT_LO);
    if(number_channels > 2) {
```

```
    tpu_hsr(tpu, param2_channel, TPU_TSM_INIT_LO);
     }
    if(number_channels > 3) {
    tpu_hsr(tpu, param3_channel, TPU_TSM_INIT_LO);
     }



    /* Enable channels by assigning a priority to them. */
    /* All Channels MUST have the same priority.       */
    tpu_enable(tpu, master_channel, priority);
    tpu_enable(tpu, param1_channel, priority);
    tpu_enable(tpu, param2_channel, priority);
    tpu_enable(tpu, param3_channel, priority);

}  /* End tpu_tsm_init */

/****************************************************/
/*    End of TPU Initialization                     */
/****************************************************/

/**************************************************/
/* Generate other tasks for the TSM function.     */
/**************************************************/

/*******************************************************************************
FUNCTION      : tpu_tsm_mov
PURPOSE       : To activate the move and acceleration of the stepper motor.
INPUTS NOTES  : This function has 3 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                         type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the number of the master channel
                 position - The new desired position for the stepper motor.
GENERAL NOTES : The channel must be a master channel for the TSM function. There
                 can be more than one master channel in the same TPU. The
                 additional master channel(s) would be for another TSM function.
*******************************************************************************/

void tpu_tsm_mov(struct TPU3_tag *tpu, UINT8 channel, UINT16 position)
    {
/* write a new desired position into the TSM master channel */
    tpu->PARM.R[channel][TPU_TSM_DESIRED_POSITION] = position;
```

```
/* Issue a move request to the host service request (master channel:HSR=0x11) */
    tpu_hsr(tpu, channel, TPU_TSM_HSR_MOV);
    };

/********************************************************************************
FUNCTION      : tpu_tsm_rd_dp
PURPOSE       : To read the desired position.
INPUTS NOTES  : This function has 2 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                         type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the number of the master channel
RETURN NOTES  : desired position - The desired position is returned as a UINT16.
GENERAL NOTES : The channel must be a master channel for the TSM function. The
                 read of the desired position can be used to drive the program.
********************************************************************************/

UINT16 tpu_tsm_rd_dp(struct TPU3_tag *tpu, UINT8 channel)
    {
/* read and return the desired position from the TSM master channel */
    return (tpu->PARM.R[channel][TPU_TSM_DESIRED_POSITION]);
    };

/********************************************************************************
FUNCTION      : tpu_tsm_rd_cp
PURPOSE       : To read the current position.
INPUTS NOTES  : This function has 2 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                         type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the number of the master channel
RETURN NOTES  : current position - The desired position is returned as a UINT16.
GENERAL NOTES : The channel must be a master channel for the TSM function. The
                 read of the current position can be used to drive the program.
********************************************************************************/

UINT16 tpu_tsm_rd_cp(struct TPU3_tag *tpu, UINT8 channel)
    {
/* read and return the current position from the TSM master channel */
    return (tpu->PARM.R[channel][TPU_TSM_CURRENT_POSITION]);
    };
```

```
/***************************************************/
/* Generate Interrupt tasks for the TSM interrupt. */
/***************************************************/
/*******************************************************************************
FUNCTION      : tpu_tsm_int_lev
PURPOSE       : To program the level of interrupt from the TPU-TSM function.
INPUTS NOTES  : This function has 2 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                         type TPU3_tag which is defined in m_tpu3.h
                 level - The interrupt level (0 to 31).
GENERAL NOTES : Level must be in the range of 0 to 31.
*******************************************************************************/
void tpu_tsm_int_lev(struct TPU3_tag *tpu, UINT8 level)
   {
     int remainder;
          if( level > 23) {
            tpu->TICR.B.ILBS = 0x11;
            remainder = level - 24;
            tpu->TICR.B.CIRL = remainder;
          }
          else if( level > 15) {
            tpu->TICR.B.ILBS = 0x10;
            remainder = level - 16;
            tpu->TICR.B.CIRL = remainder;
          }
          else if( level > 7) {
            tpu->TICR.B.ILBS = 0x01;
            remainder = level - 8;
            tpu->TICR.B.CIRL = remainder;
          }
          else {
            tpu->TICR.B.ILBS = 0x00;
            tpu->TICR.B.CIRL = level;
          }
   };

/*******************************************************************************
FUNCTION      : tpu_tsm_int_chk
PURPOSE       : Interrupt check for TSM interrupt service request match.
INPUTS NOTES  : This function has 3 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
```

```
                         type TPU3_tag which is defined in m_tpu3.h
                 channel - The master channel.
RETURNS       : Returns the value TRUE (1) or FALSE (0) as a UINT8.
GENERAL NOTES : Level must be incoded according to the SIPEND register.
*******************************************************************************/

int tpu_tsm_int_chk(struct TPU3_tag *tpu, UINT16 channel) {
    /* Determine that TPU was the source */
    /* Determine the tpu channel interrupt source */
  int val;
  if (tpu->CISR.R == channel) {
         val = TPU_TSM_TRUE;
       }
   else val = TPU_TSM_FALSE;
 return val;
}

/*******************************************************************************
FUNCTION      : tpu_tsm_cisr_clr
PURPOSE       : Interrupt clear for TSM cisr register.
INPUTS NOTES  : This function has 1 parameter:
                 level - The interrupt level (0 to 7) as encoded through the
                         32-bit SIPEND register.
GENERAL NOTES : Level must be incoded according to the SIPEND register. This
                 level is 32-bits long (UINT32).
*******************************************************************************/

void tpu_tsm_cisr_clr(struct TPU3_tag *tpu, UINT16 CISR_level) {
    /* Write a "1" to the active pending bit to reset it.   */
    /* A logic bit-wise "OR" between the SIPEND and the    */
    /* level will clear the specific SIPEND level.         */

/* Clear any existing TPU interrupts    */
   tpu->CISR.R = 0x0000;
 /*  tpu->CISR.R &= ~(1 << CISR_level);  */
}

/*******************************************************************************
FUNCTION      : tpu_tsm_mas_chan_cier
PURPOSE       : Convert master channel integer into CIER, CISR UINT16 value
INPUTS NOTES  : This function has 1 parameter:
```

```
                 master_chan - The channel (0 to 15) to convert to encodeding
                 for the 16-bit TPU CIER, or CISR register.
GENERAL NOTES : master_chan integer input is returned as a UINT16 value
                 for the TPU CIER and CISR registers.
*******************************************************************************/

UINT16 tpu_tsm_mas_chan_cier(int master_chan) {
    /* Convert integer input master channel to the  */
    /* TPU CIER or CISR encoding.                   */

UINT16 cier_chan = 0x0000;
   if(master_chan == 0)  { cier_chan = 0x0001; }
   if(master_chan == 1)  { cier_chan = 0x0002; }
   if(master_chan == 2)  { cier_chan = 0x0004; }
   if(master_chan == 3)  { cier_chan = 0x0008; }
   if(master_chan == 4)  { cier_chan = 0x0010; }
   if(master_chan == 5)  { cier_chan = 0x0020; }
   if(master_chan == 6)  { cier_chan = 0x0040; }
   if(master_chan == 7)  { cier_chan = 0x0080; }
   if(master_chan == 8)  { cier_chan = 0x0100; }
   if(master_chan == 9)  { cier_chan = 0x0200; }
   if(master_chan == 10) { cier_chan = 0x0400; }
   if(master_chan == 11) { cier_chan = 0x0800; }
   if(master_chan == 12) { cier_chan = 0x1000; }
   if(master_chan == 13) { cier_chan = 0x2000; }
   if(master_chan == 14) { cier_chan = 0x4000; }
   if(master_chan == 15) { cier_chan = 0x8000; }

   return cier_chan;
}


/******************************************************************************/
/* Freescale reserves the right to make changes without further notice to any */
/* product herein to improve reliability, function, or design. Freescale does */
/* not assume any  liability arising  out  of the  application or use of any */
/* product,  circuit, or software described herein;  neither  does it convey */
/* any license under its patent rights  nor the  rights of others.  Freescale */
/* products are not designed, intended,  or authorized for use as components */
/* in  systems  intended  for  surgical  implant  into  the  body, or  other */
/* applications intended to support life, or  for any  other application  in */
/* which the failure of the Freescale product  could create a situation where */
/* personal injury or death may occur. Should Buyer purchase or use Freescale */
/* products for any such intended  or unauthorized  application, Buyer shall */
/* indemnify and  hold  Freescale  and its officers, employees, subsidiaries, */
/* affiliates,  and distributors harmless against all claims costs, damages, */
/* and expenses, and reasonable  attorney  fees arising  out of, directly or */
/* indirectly,  any claim of personal injury  or death  associated with such */
/* unintended or unauthorized use, even if such claim alleges that  Freescale */
/* was negligent regarding the  design  or manufacture of the part. Freescale */
/* and the Freescale logo* are registered trademarks of Freescale, Inc..         */
/*****************************************************************************/
```

10.2 tpu_tsm.h

```
          /**************************************************************************/
/* FILE NAME: tpu_tsm.h                       COPYRIGHT (c)  2002 */
/* VERSION:  0.1                                  All Rights Reserved     */
/*                                                                        */
//* DESCRIPTION:                                                          */
/* This file defines all of the registers and bit fields on the TPU3      */
/* Table Stepper Motor (TSM) function.                                    */
/*========================================================================*/
/* AUTHOR: Glenn Jackson                                                  */
/* COMPILER: Diab Data        VERSION: 4.3g                               */
/*                                                                        */
/* HISTORY                                                                */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                  */
/* ---   -----------  ---------    ---------------------                  */
/* 0.1   G. Jackson   30/Jul/02    Initial version of file for            */
/*                                   Spanish Oak.                         */
/**************************************************************************/


#include "m_tpu3.h"


#ifdef  __cplusplus
extern "C" {
#endif


/*************************************************************/
/*            Definition of terms and initial settings       */
/*************************************************************/
```

```
/* Define the maximum parameter table size */
/*#define TABLE_X 3       Table x value        */
/*#define TABLE_Y 8       Table y value        */

/* Define HSQ values */
#define TPU_TSM_LOCAL_ACC_TBL 0x0
#define TPU_TSM_SPLIT_ACC_TBL 0x1
#define TPU_TSM_ROTATE_ONCE 0x0
#define TPU_TSM_ROTATE_TWICE 0x2

/* Define HSR values */
#define TPU_TSM_NO_HOST 0x0
#define TPU_TSM_INIT_LO 0x1
#define TPU_TSM_INIT_HI 0x2
#define TPU_TSM_HSR_MOV 0x3

/* Define pin state */
#define TPU_TSM_PIN_HIGH 0x8000
#define TPU_TSM_PIN_LOW  0x0000

/* Define test result values */
#define TPU_TSM_TRUE  1
#define TPU_TSM_FALSE 0

/* Define parameter RAM locations */
#define TPU_TSM_DESIRED_POSITION   0
#define TPU_TSM_CURRENT_POSITION   1
#define TPU_TSM_TABLE              2
#define TPU_TSM_TABLE_SIZE         2
#define TPU_TSM_TABLE_INDEX        2
#define TPU_TSM_SLEW_PERIOD        3
#define TPU_TSM_S                  3
#define TPU_TSM_START_PERIOD       4
#define TPU_TSM_A                  4
#define TPU_TSM_PIN_SEQUENCE       5

/* Define interrupt levels   */
/* Define the USIU.SIPEND level encodings */
/*
#define TSM_INT_LEVEL0 0x40000000
```

```
#define TSM_INT_LEVEL1 0x10000000
#define TSM_INT_LEVEL2 0x04000000
#define TSM_INT_LEVEL3 0x01000000
#define TSM_INT_LEVEL4 0x00400000
#define TSM_INT_LEVEL5 0x00100000
#define TSM_INT_LEVEL6 0x00040000
#define TSM_INT_LEVEL7 0x00010000
*/

/* Define TPU.CISR interrupt channel encodings   */
/*
#define TSM_CISR_INT_CHANNEL0  0x0001
#define TSM_CISR_INT_CHANNEL1  0x0002
#define TSM_CISR_INT_CHANNEL2  0x0004
#define TSM_CISR_INT_CHANNEL3  0x0008
#define TSM_CISR_INT_CHANNEL4  0x0010
#define TSM_CISR_INT_CHANNEL5  0x0020
#define TSM_CISR_INT_CHANNEL6  0x0040
#define TSM_CISR_INT_CHANNEL7  0x0080
#define TSM_CISR_INT_CHANNEL8  0x0100
#define TSM_CISR_INT_CHANNEL9  0x0200
#define TSM_CISR_INT_CHANNEL10 0x0400
#define TSM_CISR_INT_CHANNEL11 0x0800
#define TSM_CISR_INT_CHANNEL12 0x1000
#define TSM_CISR_INT_CHANNEL13 0x2000
#define TSM_CISR_INT_CHANNEL14 0x4000
#define TSM_CISR_INT_CHANNEL15 0x8000
*/
/* Prototype of functions */

void tpu_tsm_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority,
                  INT16 start_position, UINT16 table_size_index,
                  UINT16 slew_period, UINT16 start_period,
                  UINT16 pin_sequence, UINT8 number_channels,
                  UINT16 *table, UINT8 table_size);
void tpu_tsm_mov(struct TPU3_tag *tpu, UINT8 channel, UINT16 position);
UINT16 tpu_tsm_rd_dp(struct TPU3_tag *tpu, UINT8 channel);
UINT16 tpu_tsm_rd_cp(struct TPU3_tag *tpu, UINT8 channel);

UINT16 tpu_tsm_mas_chan_cier(int master_chan);
```

```
void tpu_tsm_int_lev(struct TPU3_tag *tpu, UINT8 level);
int tpu_tsm_int_chk(struct TPU3_tag *tpu, UINT16 channel);
void tpu_tsm_cisr_clr(struct TPU3_tag *tpu, UINT16 CISR_level);

#ifdef  __cplusplus
}
#endif
```

10.3 tpu_tsm_ex1.c

The developer should use this code as an example for calling the TPU TSM functions and incorporating these calls into a final application.

```
/**************************************************************************/
/* FILE NAME: tpu_tsm_ex1.c                   COPYRIGHT (c)  2002 */
/* VERSION: 1.0                                   All Rights Reserved     */
/*                                                                        */
/* DESCRIPTION: This sample program shows a simple example of a program   */
/* that initializes the TSM function and updates a desired position       */
/* varible from the CPU. The position in the current position moves most  */
/* efficiently to the new desired position. The table of byte variables   */
/* establishes the rate of speed. Advancing through the next byte step in */
/* the table will accelerate the speed of the motor. (ie. Table Stepper   */
/* Motor--TSM).                                                           */
/* The example uses the TPU A on a master channel and up to three         */
/* consecutive channels. The master channel for this example is 0. With   */
/* channels 1, 2, and 3 as the table parameter channels.                  */
/* The program is targeted for the MPC555 but should work on any MPC500   */
/* device with a TPU. For other devices the setup routines will also need */
/* to be changed.                                                         */
/*========================================================================*/
/* HISTORY           ORIGINAL AUTHOR: Jeff Loeliger                       */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                  */
/* ---   -----------  ---------    ---------------------                  */
/* 1.0   J. Loeliger  03/Aug/02    Initial version of function.           */
/* 1.1   G. Jackson   12/Aug/02    Convert to TSM example.                */
/**************************************************************************/

#include "mpc565.h"     /* Define all of the MPC555 registers, this needs to */
                        /* changed if other MPC500 devices are used.         */
```

```
#include "mpc500.c"     /* Configuration routines for MPC555 EVB, will need */
                        /* to be changed if other hardware is used.         */
#include "mpc500_util.h"    /* Utility routines for using MPC500 devices    */
#include "tpu_tsm.h"    /* TPU TSM functions */


/* Prototypes for functions of example program */
void Ext_Isr(void);
void tpu_tsm_int_config(struct TPU3_tag *tpu, UINT8 channel, UINT8 level);
UINT32 tpu_tsm_sip_int_lvl(int int_level);


/* Initialization parameters -- redefine these parameters for a      */
/*                              second parallel TSM initialization.  */
UINT8 mas_channel = 13;  /* master channel designation, changes once here. */
UINT8 int_level = 2;     /* Set global interrupt level here.               */
UINT8 num_channel = 4;   /* Number of channels.                            */
UINT8 table_size = 24;   /* Number of bytes/2 in the parameter table.      */
INT16 start_position = 0x0010; /* Initial value for desired_ & current_position */
UINT16 table_siz_index = 0x1C00; /* Initial value for table size(1C) and index(00) */
UINT8  flag = 0;           /* Interrupt test flag.                        */


     struct TPU3_tag *tpua = &TPU_A;   /* pointer for TPU routines */


void main ()
{
     int x;  /* Just an integer to hold a value  */
  UINT16 dp_val; /* Value of desired position for program control. */
  UINT16 cp_val; /* Value of current position for program control. */


/* An example of a parameter table is shown below. Change this table */
/* to meet your requirements. Duplicate this table with another      */
/* name for a second TSM function (third, etc.), if different.       */


static UINT16 table[] = {0xF7FF, 0xE7F0, 0xD7E0, 0xC7D0,
                         0xB7C0, 0xA7B0, 0x97A0, 0x8790,
                         0x7780, 0x6770, 0x5760, 0x4750,
                         0x3740, 0x2730, 0x1720, 0x0F10,
                         0x0E0F, 0x0C0D, 0x0A0B, 0x0809,
                         0x0607, 0x0405, 0x0203, 0x0001};


/* Hardware Setup -- machine settings (watchdog, timers, speed, etc.) */
     setup_mpc500(40);       /*Setup device and programm PLL to 40MHz*/
```

```
    /* Initialize Table Stepper Motor function with:         */
    /*    -Input signal on TPU A,                            */
    /*    -master_channel = mas_channel                      */
    /*    -Schedule as high priority in the TPU              */
    /*    -Initial desired position of 0x0010,                */
    /*    -Initial current position of 0x0010,                */
    /*    -Table Size of 0x18 (d24), Table Index of 0x00,    */
    /*    -Slew Period of 0x2000 (shift to 0x4000),  S = 0,  */
    /*    -Start Period of 0x6800 (shift to 0xD000), A = 1,  */
    /*    -Pin Sequence of 0xE0E0.                           */
    /*    -Number of channels = 4.                           */
    /*    -*table -- start address of parameter table.       */
    /*    -load_table_size -- number of table bytes to load. */

 tpu_tsm_init(tpua, mas_channel, TPU_PRIORITY_HIGH, start_position,
              table_siz_index, 0x4000, 0xD001, 0xE0E0, num_channel,
              table, table_size);

/* Enable interrupts; set the interrupt level in the TPU CIER   */
     tpu_tsm_int_config(tpua, mas_channel, int_level);

/**************************************************/
/*    Generate a new desired position.            */
/**************************************************/

/* Designate which tpu (A), the master channel = mas_channel,        */
/*      and the new desired location value (0x3000)                  */
/* NOTE: The Master Channel cannot change from the init designation! */
    dp_val = 0x3000;

      tpu_tsm_mov(tpua, mas_channel, dp_val);

/**************************************************/
/* Make changes after an interrupt.               */
/**************************************************/

   while (!flag ) { /* Interrupt routine will set this flag */
/* Do other productive work while waiting for move to complete */
       x=8;
    };

/* Interrupt has occurred -- continue:  */
/*  Below are examples of control logic used in a TSM function. */
/*  Adapt this to your specific needs and controls.             */
     dp_val = tpu_tsm_rd_dp(tpua, mas_channel);
     cp_val = tpu_tsm_rd_cp(tpua, mas_channel);
     if(dp_val = cp_val) { /* poll to see if current position has         */
                           /* reached desired position. Set new position. */
/* When return from interrupt, set a new desired location */
        if(cp_val > 2000) {
           dp_val -= 0x1000;
           tpu_tsm_mov(tpua, mas_channel, dp_val);
          }
        else {
               dp_val += 0x0500;
               tpu_tsm_mov(tpua, mas_channel, dp_val);
             }
       }


   while(1){
      /* Hold at end of program */
       x=4;
     };
}          /* End of main */

/*******************************************************************************
FUNCTION      : Ext_Isr
PURPOSE       : Interrupt call at interrupt request.
INPUTS NOTES  : This function has 0 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                         type TPU3_tag which is defined in m_tpu3.h
                 level - The interrupt level (0 to 31).
GENERAL NOTES : This routine is org'ed at 0x500. Only level 2 is shown as an
                 example here.
*******************************************************************************/
/* Below is a Diab Data centric routine. Other compilers */
/*  may require a different structure for an external    */
/*  interrupt service request.                           */
```

```
void Ext_Isr()
{
  UINT16 mas_chan_cisr;
       mas_chan_cisr = tpu_tsm_mas_chan_cier(mas_channel);
 /* Check that the level of interrupt matches the TPU, TSM .*/
       flag = (tpu_tsm_int_chk(tpua, mas_chan_cisr));
  /* Force the clearance of the CISR register */
       tpu_tsm_cisr_clr(tpua, mas_chan_cisr);

}

/********************************************************************************
FUNCTION      : tpu_tsm_int_config
PURPOSE       : Interrupt activities when TSM interrupt request occurs.
INPUTS NOTES  : This function has 2 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                         type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the number of the master channel
                 level - The interrupt level (0 to 7).
GENERAL NOTES : Channel must be a TSM master channel. Level must be in the
                 range of 0 to 31. The activities listed here are for purposes
                 of example and to use all of the functions once.
********************************************************************************/

void tpu_tsm_int_config(struct TPU3_tag *tpu, UINT8 channel, UINT8 level) {
     /* Shut down CPU MSR interrupts */
   UINT16 cisr_chan;
   UINT32 sipend_int_level;

    asm (" mtspr EID, r0 ");  /*  MSR[EE] = 0; MSR[RI] = 1 */

    cisr_chan = tpu_tsm_mas_chan_cier(channel);
    sipend_int_level = tpu_tsm_sip_int_lvl(level);
    /* Clear current interrupt requests              */
  /* Force the clearance of the CISR register */
    tpu_tsm_cisr_clr(tpua, cisr_chan);

    /* Configure the TPU Interrupt Configuration Register (TICR)  */
    /*  CIRL=2, ILBS=0 for a level 2 interrupt.                   */
    tpu_tsm_int_lev( tpu, level);
    /* Set interupt enable to master channel CIER = master channel */
    tpu_interrupt_enable( tpu, channel);

    /* Set the USIU mask to accept a level 2 interrupt */
    USIU.SIMASK.R = sipend_int_level;

  /* Force the clearance of the CISR register */
    tpu_tsm_cisr_clr(tpua, cisr_chan);

/* Set the CPU MSR to enable interrupts. EIE activates MSR[EE], MSR[RI] */
   asm(" mtspr EIE, r3");               /* NOTE: r3 is a moot register */
   };

/*******************************************************************************
FUNCTION      : tpu_tsm_sip_int_lvl
PURPOSE       : Convert interrupt level integer into USIU.SIPEND UINT32 value
INPUTS NOTES  : This function has 1 parameter:
                 level - The interrupt level (0 to 7) as encoded through the
                         32-bit SIPEND register.
GENERAL NOTES : master_chan integer input is returned as a UINT16 value
                 for the TPU CIER and CISR registers.
*******************************************************************************/


UINT32 tpu_tsm_sip_int_lvl(int int_level) {
    /* Convert integer input interrupt level  */
    /* to the 32-bit SIPEND encoding.         */

UINT32 sip_level = 0x00000000;
   if(int_level == 0)  { sip_level = 0x40000000; }
   if(int_level == 1)  { sip_level = 0x10000000; }
   if(int_level == 2)  { sip_level = 0x04000000; }
   if(int_level == 3)  { sip_level = 0x01000000; }
   if(int_level == 4)  { sip_level = 0x00400000; }
   if(int_level == 5)  { sip_level = 0x00100000; }
   if(int_level == 6)  { sip_level = 0x00040000; }
   if(int_level == 7)  { sip_level = 0x00010000; }

   return sip_level;
}
```

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

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.
Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.