Freescale Semiconductor

Order this document
by TPUPN09/D

PROGRAMMING NOTE

Multiphase Motor Commutation TPU Function (COMM)

by Jeff Wright

1 Functional Overview

The commutation (COMM) TPU function uses multiple TPU channels to produce the drive enable signals necessary for commutating brushless motors. Motor types supported include three- and four-phase brushless dc and three-phase switched reluctance motors. The COMM function is used in conjunction with another TPU function to provide a choice of sensored (Hall effect or optical) or sensorless (from an encoder) commutation. The signals produced by COMM are externally gated with a PWM, also generated by the TPU, to drive the motor. The COMM function has been optimized for flexibility and this may allow it to meet the requirements of other multisignal applications.

2 Detailed Description

The COMM TPU function produces the commutation drive signals for a variety of brushless motor types. The function can use up to eight adjacent TPU channels (one master channel and from one to seven slave channels) to generate the commutation signals. For example, six channels can be used for a three-phase brushless dc motor or four channels can be used for a four-phase motor. All TPU service activity takes place on the master channel — the slaves are used only as synchronized output pins. When a commutation state change occurs, the pin state change on each output channel is synchronized using an output match based on the TCR1 counter.

COMM has been designed to work with several TPU input functions to provide a choice of sensored or sensorless commutation. To support these input functions, COMM has two basic operating modes selected by the host sequence bits on the master channel:

Sensored Mode is designed to work with the Hall effect decode (HALLD) TPU function. HALLD is an input function that decodes either two or three Hall effect or optical sensors and a CPU-supplied direction input into a state number. HALLD writes this state number into parameter RAM of the COMM function master channel and then issues a link to that channel. On receipt of the link, COMM obtains the output pin configuration that corresponds to the new commutation state from a table in parameter RAM and subsequently outputs the new commutation state. The user has complete control over the commutation sequence via the programmable state table in the COMM function and a CPU force feature which allows the CPU to force a particular commutation state. **Figure 1** shows a typical Hall effect setup and control waveforms.

Sensorless Mode is designed to support sensorless commutation from a high resolution encoder on the shaft of a motor. To keep the COMM function as flexible as possible, the sensorless mode of operation has been implemented as a programmable state machine. The basis for the production of the commutation signals is a counter representing the angular position of the motor. This counter is the position count output of a TPU input function such as quadrature decode (QDEC), or fast quadrature decode (FQD), which derives angular position from the shaft encoder. **Figure 2** shows typical shaft encoder setup and control waveforms.

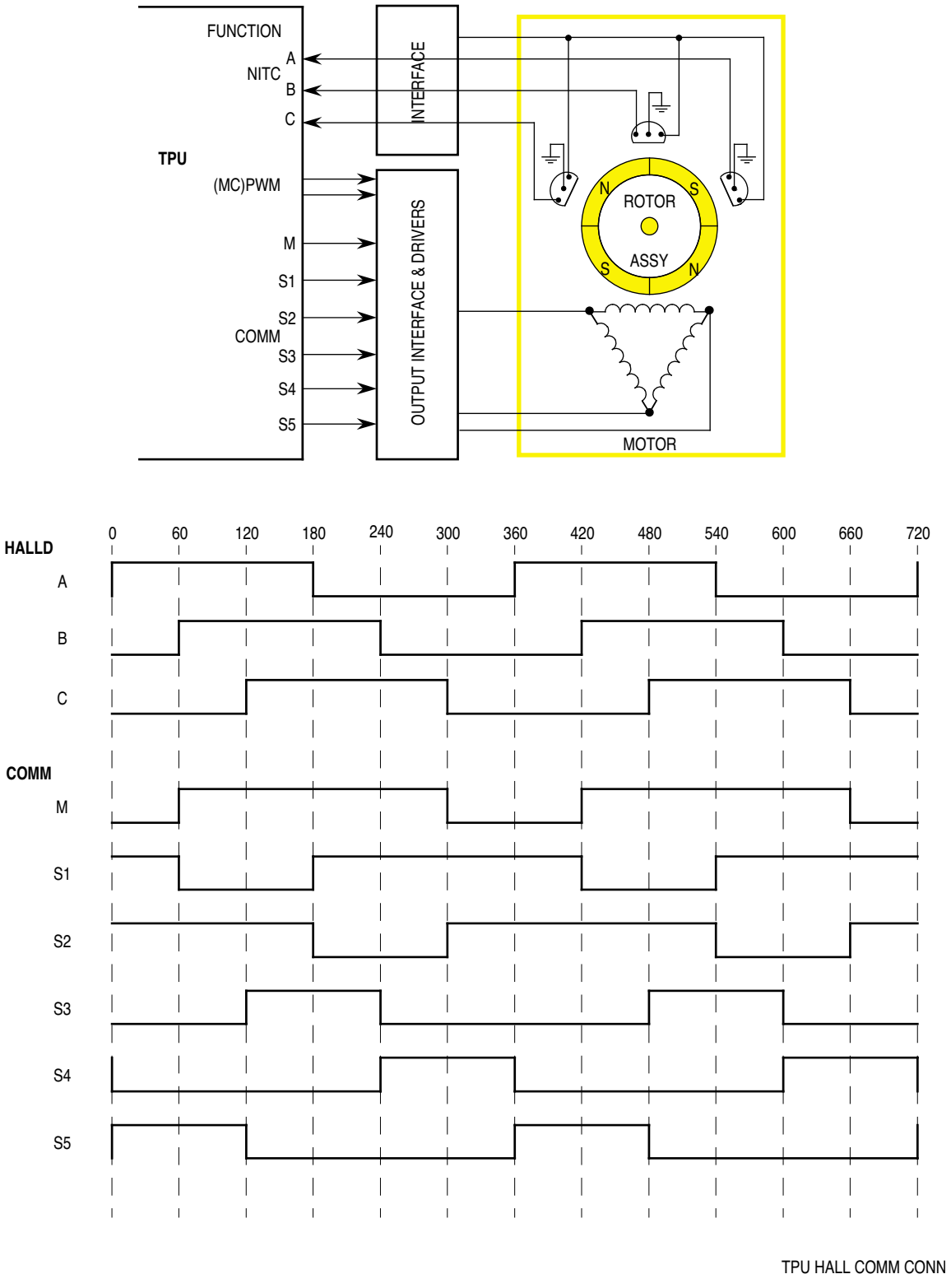

Figure 1 Typical Hall Effect Setup and Waveforms

The COMM function directly accesses this counter, located anywhere in parameter RAM, and performs tests on it to determine the required state of the commutation output pins. This process is carried out without CPU intervention. The COMM function maintains upper and lower angular limits to the current state in parameter RAM. On each service of the COMM function, the position counter is compared with these limits. If the position count has passed either limit, the state number is updated and a new state parameter is obtained from a circular table in parameter RAM. The state parameter contains pin configuration and length of the new state in position counts. New upper and lower angular limits are calculated using the length of the new state, then stored for use in subsequent state tests; the new pin configuration is subsequently output on the COMM channels.

As an additional feature of sensorless operation, a CPU supplied angular offset is added to the position count before the limit tests. This parameter, which can be updated at any time, allows the CPU to advance or retard previously programmed state switching angles. The offset parameter can be used to start the motor in a particular direction, to partially compensate for TPU service latencies, to maintain torque at high motor speeds, and to force braking on the motor. See **7 Performance and Use of Function** for more detail.

The number of states in the commutation sequence, the length in angular position counts of each state, the number of channels used for commutation, and the pin states for each state are all independently programmable by the user. These capabilities make COMM suitable for a wide variety of commutation schemes.

The size of the state table determines the number of states that can be generated. Maximum table size is dependent upon which TPU channel is picked as the master COMM channel. For channels 0 to 12 and channel 15, six states can be implemented; for channel 13, up to 22 states are possible, and 14 states can be implemented if channel 14 is chosen as the master channel. Up to eight TPU channels can be used as COMM signal outputs (including the master channel), and each state has a length which is individually programmable over an 8-bit range in position counts. In applications using a very high resolution encoder, an 8-bit range for the state length in position counts may not be sufficient. COMM allows the use of multiple state table entries, programmed to have the same output pin configuration, to effectively lengthen the state.

In sensorless operation, the COMM function has been designed to update the commutation signals on a periodic basis. There are two “sub-modes” of sensorless operation:

In **Match Update mode**, a user programmable periodic match on the master COMM channel is used to invoke updating of the commutation signals. Assuming an accurate position encoder, the accuracy of commutation is dependent on this periodic update rate. A faster update rate results in more accurate commutation, but higher TPU overhead. This mode is provided for use with the QDEC and FQD TPU input functions.

In **Link Update mode**, the update of the commutation signals is invoked by a link request from another TPU channel. Assuming an accurate position encoder, the frequency of the link from the other channel determines the accuracy of commutation. A faster update rate results in more accurate commutation, but higher TPU overhead. This mode is provided for use with the DUC TPU function.

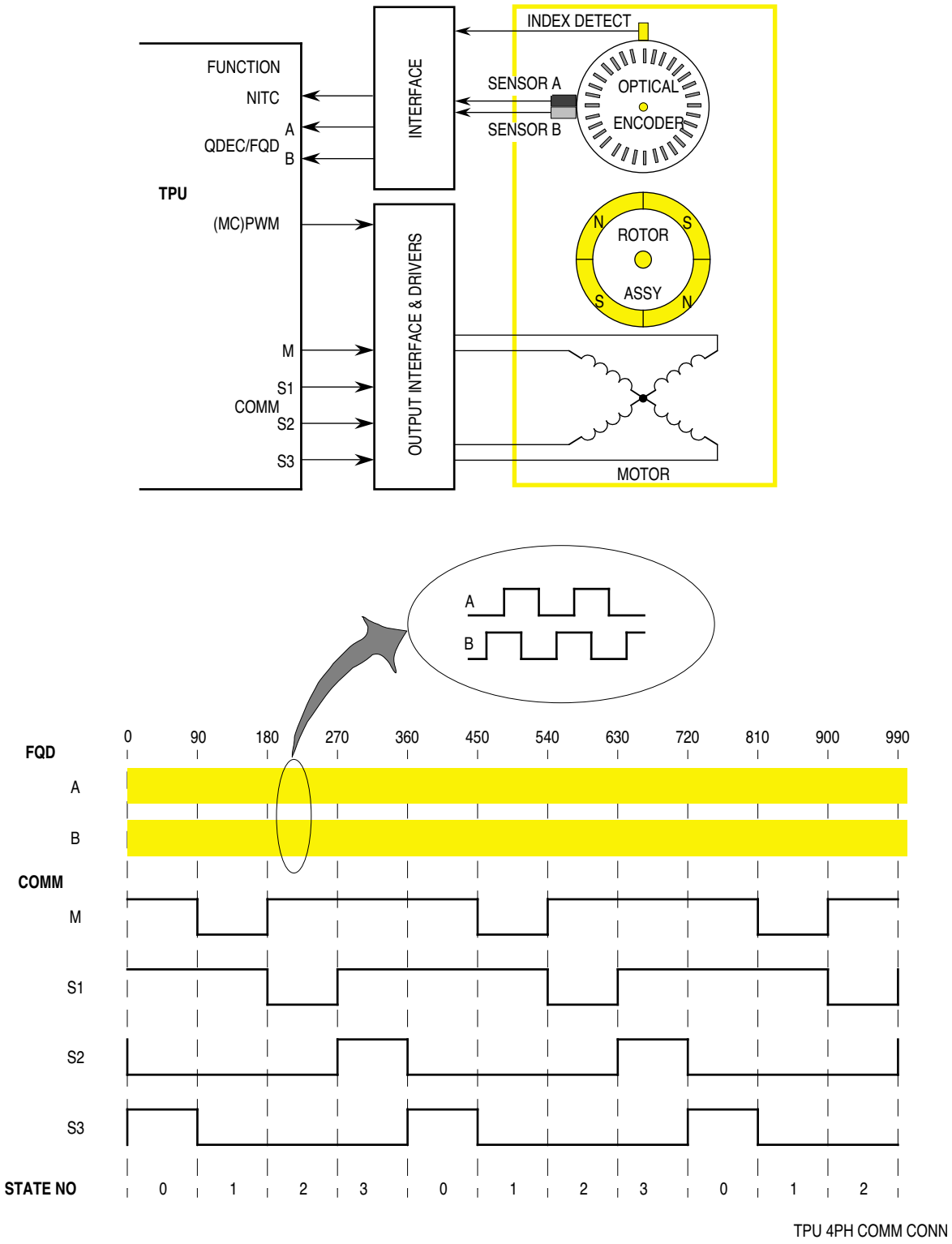

Figure 2 Typical Sensorless Setup and Waveforms

2.1 All Modes

The host CPU can force any commutation state, to put the motor stator field into a known configuration, at any time. The CPU can also interrogate the COMM function at any time to determine which state is currently active. Since the state table is programmed during initialization, the user can decide which state number corresponds to which output pin configuration. This flexibility allows a variety of commutation schemes to be implemented.

To drive the motor, the outputs of the COMM function TPU channels are used to gate a PWM generated on another TPU channel onto the motor phase drivers.

The TPU is flexible enough to support an application with both Hall effect or optical sensors for commutation and an encoder on the motor shaft for deriving speed and direction information. This may be beneficial in environments where encoder information cannot be relied upon to be completely accurate.

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. COMM function code size is:

42 μ instructions + 8 entries = **50 long words**

4 Function Parameters

This section provides detailed descriptions of function parameters stored in channel parameter RAM. **Figure 3** shows TPU parameter RAM address mapping. **Figure 4** shows the parameter RAM assignment used by the function. In the diagrams, Y = M111, where M is the value of the module mapping bit (MM) in the system integration module configuration register (Y = $7 or $F).

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

— = Not Implemented (reads as $00)

Figure 3 TPU Channel Parameter RAM CPU Address Map

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$YFFFW0					NO_OF_PINS				COUNTER_ADDR							
$YFFFW2	NO_OF_STATES								STATE_NO							
$YFFFW4	OFFSET															
$YFFFW6	UPDATE_PERIOD															
$YFFFW8	UPPER															
$YFFFWA	LOWER															
$YFFFWC																
$YFFFWE																

	Commutation Sequence Parameters		
$YFFF(W+1)0	(LENGTH)	STATE 0	(PIN CONFIG)
$YFFF(W+1)2	(LENGTH)	STATE 1	(PIN CONFIG)
$YFFF(W+1)4	(LENGTH)	STATE 2	(PIN CONFIG)
$YFFF(W+1)6	(LENGTH)	STATE 3	(PIN CONFIG)
$YFFF(W+1)8	(LENGTH)	STATE 4	(PIN CONFIG)
$YFFF(W+1)A	(LENGTH)	STATE 5	(PIN CONFIG)
$YFFF(W+1)C	(LENGTH)	STATE 6*	(PIN CONFIG)
$YFFF(W+1)E	(LENGTH)	STATE 7*	(PIN CONFIG)
$YFFF(W+2)0	(LENGTH)	STATE 8*	(PIN CONFIG)
$YFFF(W+2)2	(LENGTH)	STATE 9*	(PIN CONFIG)
$YFFF(W+2)4	(LENGTH)	STATE 10*	(PIN CONFIG)
$YFFF(W+2)6	(LENGTH)	STATE 11*	(PIN CONFIG)
$YFFF(W+2)8	(LENGTH)	STATE 12*	(PIN CONFIG)
$YFFF(W+2)A	(LENGTH)	STATE 13*	(PIN CONFIG)
$YFFF(W+2)C	(LENGTH)	STATE 14*	(PIN CONFIG)
$YFFF(W+2)E	(LENGTH)	STATE 15*	(PIN CONFIG)
$YFFF(W+3)0	(LENGTH)	STATE 16*	(PIN CONFIG)
$YFFF(W+3)2	(LENGTH)	STATE 17*	(PIN CONFIG)
$YFFF(W+3)4	(LENGTH)	STATE 18*	(PIN CONFIG)
$YFFF(W+3)6	(LENGTH)	STATE 19*	(PIN CONFIG)
$YFFF(W+3)8	(LENGTH)	STATE 20*	(PIN CONFIG)
$YFFF(W+3)A	(LENGTH)	STATE 21*	(PIN CONFIG)

W = Master COMM channel number

* These parameters are not available in all configurations — see text.

Parameter Write Access

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

Figure 4 Comm Function Parameter RAM Assignment

4.1 COUNTER_ADDR

This 8-bit parameter is initialized by the CPU to contain the address in parameter RAM of the angular position counter (POSITION_COUNT) that is used as the basis for the state tests in sensorless mode. For instance, if POSITION_COUNT is resident in the second parameter of channel 4, $42 is stored in COUNTER_ADDR. COUNTER_ADDR must not be changed while the motor is running. COUNTER_ADDR is not used in sensored mode.

4.2 NO_OF_PINS

This CPU-written parameter contains the number of TPU channels used to generate commutation signals (including the master channel). Although this is a 4-bit parameter, the maximum number of channels that can be used is eight, corresponding to the length of the PIN_CONFIG field of STATE_N. To use six channels for three-phase commutation, enter $06 in NO_OF_PINS. The valid range for NO_OF_PINS is $1 \leq$ NO_OF_PINS ≤ 8. NO_OF_PINS must not be changed while the motor is running.

4.3 STATE_NO

This 8-bit parameter is used by the TPU to keep track of the current commutation state and it is used as an address to obtain the correct state parameter from the table. STATE_NO can be read at any time by the CPU. In sensorless mode, STATE_NO is updated by the TPU when the COMM function is serviced and a state change results. In sensored mode, the HALLD TPU function writes a state number into STATE_NO before issuing a link request to the COMM channel. STATE_NO may not reflect the commutation state currently in effect due to the delay between the update of STATE_NO and the update of the COMM function channel pins (dependent on UPDATE_PERIOD and TPU latency).

The CPU can also write STATE_NO and use an HSR %10 to force the outputs to the corresponding configuration for the new state. A sequence of these force commands can be used during start-up of the motor to align it into a known condition. See **7 Performance and Use of Function** for more information.

The valid range for STATE_NO is 0 to (NO_OF_STATES –1).

4.4 NO_OF_STATES

This 8-bit CPU-written parameter determines the number of states in the commutation sequence in sensorless mode. For example, $06 must be written to NO_OF_STATES for six state commutation. The number of states which can be implemented is limited by the amount of contiguous parameter RAM available, starting with parameter 0 of the first slave channel (next higher channel in numeric order from master channel). The amount of contiguous parameter RAM is dependent upon which channel is chosen as the master, due to unimplemented parameter RAM locations (See **Figure 3**) — for channels 0 to 12 and channel 15, six states can be implemented; for channel 13, up to 22 states are possible, and 14 states can be implemented if channel 14 is chosen as master. NO_OF_STATES must be greater than zero at all times. When using COMM in sensored mode with the HALLD function, the NO_OF_STATES parameter is unused — it is overwritten when HALLD writes a new STATE_NO.

4.5 OFFSET

In sensorless mode, this 16-bit signed parameter can be used by the host CPU to advance or retard all the state switching angles on the fly. OFFSET is added to the position counter obtained via COUNTER_ADDR and the sum is used in position limit tests. When not used, OFFSET should be initialized to zero by the CPU. OFFSET can be written at any time; it is unused in sensored mode. See **7.5 Using Offset in Sensorless Mode** for a detailed discussion of the OFFSET parameter.

4.6 UPDATE_PERIOD

This 16-bit parameter can be used in two different ways, depending on operating mode. The valid range for UPDATE_PERIOD is 1 to $8000 (in TCR1 counts), but in practice the minimum value is higher due to TPU service latency. See **7 Performance and Use of Function** for more information.

When used for **Match Update** in sensorless mode, UPDATE_PERIOD determines the frequency of update of the commutation signals. The parameter is used to schedule a periodic match on the master channel. When the match occurs, the TPU tests the sum of the position counter and OFFSET against UPPER and LOWER and if a state change occurs, the new pin configuration appears on the output pins on the next update match. A smaller UPDATE_PERIOD results in more accurate commutation but requires greater TPU service overhead. UPDATE_PERIOD can be changed by the CPU while the motor is running to alter the accuracy of commutation at different motor speeds.

When used for **Link Update** in sensorless or sensored modes, each time a link is received by the master channel, UPDATE_PERIOD is used to schedule a match on all the COMM channels, so that the new pin states occur simultaneously on multiple channels. In this case, UPDATE_PERIOD should be programmed to a much smaller value than in match update mode, to allow the pins to change state just after completion of service of the master channel.

4.7 UPPER

This 16-bit parameter contains a value, in position counts, that corresponds to the upper angular boundary of the current commutation state. UPPER is not used in sensored mode or during a CPU force of a particular state. In sensorless modes, if the sum of OFFSET plus the position counter is greater than or equal to UPPER, STATE_NO is incremented and the pin configuration of the new state is output. After the CPU has finished forcing states during motor start-up, UPPER should be written to a value matching the current state length and position count value. Thereafter UPPER is updated automatically by the TPU when a state transition occurs.

4.8 LOWER

This 16-bit parameter contains a value, in position counts, that corresponds to the lower angular boundary of the current commutation state. LOWER is not used in sensored mode or during a CPU force of a particular state. In sensorless mode, if the sum of OFFSET plus the position counter is less than LOWER, STATE_NO is decremented and the pin configuration of the new state is output. After the CPU has finished forcing states during motor start-up, LOWER should be written to a value matching the current state length and position count value. Thereafter LOWER is updated automatically by the TPU when a state transition occurs.

4.9 USTATE_0...STATE_N

These 16-bit parameters, one for each state in the commutation sequence, reside in the parameter RAM of the slave COMM channels, starting with STATE_0 in the first parameter of slave channel 1. The maximum available number of STATE_N parameters is dependent upon the choice of master channel. Except in special cases, when channel 13 or channel 14 is selected as the master channel, the parameter RAM of the second and subsequent slave channels cannot be used for STATE_N parameters. The STATE_N parameter used at any particular time is determined by the value of STATE_NO. For instance, if STATE_NO = 3, the STATE_3 parameter is used. STATE_N parameters contain two byte-sized fields:

STATE_LENGTH specifies the length of the state in increments of the counter pointed to by COUNTER_ADDR. The length of each state is freely programmable over the range $01 to $FF unsigned. This field is not used in sensored mode.

PIN CONFIG determines the output pin levels for commutation state N. The field is right justified — if six channels are being used for commutation, the low six bits of PIN CONFIG are used to hold the pin levels.

5 Host Interface to Function

This section provides information concerning the TPU host interface to the COMM 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 = $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 5 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	Operating Mode — Only Used On Master Channel
00	Sensorless Match Update Mode
01	Sensorless Match Update Mode
10	Sensorless Link Update Mode
11	Sensored Mode

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

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

CH	Initialization
00	No Host Service (Reset Condition)
01	Not Used
10	Initialize or force state
11	Initialize or force immediate state test

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

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

CH	Channel Priority
00	Disabled
01	Low
10	Middle
11	High

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

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

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

6 Function Configuration

The steps necessary for the CPU to configure the COMM function vary according to the mode selected and the functions used to make up the rest of the drive system. These descriptions detail initialization of the COMM function only. For information concerning use and initialization of other TPU functions (MCPWM, FQD, HALLD etc.) refer to the appropriate TPU Programming Notes. Refer also to **7.2 Motor Start-Up in Sensorless Mode** and **7.3 Motor Start-Up in Sensored Mode** for more details. Initialize the COMM function as follows. (Steps 1, 2, and 3 are the same in all modes.)

6.1 Sensored Mode (HSQ[1:0] = %11)

Assumes use in conjunction with the HALLD TPU function.

1. Disable all COMM channels by clearing their two channel priority bits.
2. Select the COMM function on the selected master channel by writing the assigned COMM function number to the appropriate channel function select bits.
3. Select the desired mode of operation by writing the two host sequence bits (HSQ[0:1]) of the master COMM channel.
4. Write NO_OF_PINS and UPDATE_PERIOD.
5. Write the STATE_0 to STATE_N parameter table in slave channel parameter RAM.
6. Write a value into STATE_NO corresponding to the desired initial commutation state.
7. Issue an HSR %10 to the master channel to output the above state.
8. Write a non-zero value to the master channel priority bits to enable and prioritize service.
9. Wait for the HSR bits to be cleared by the TPU or for the interrupt status bit of the master channel to be set. This indicates the end of service. Within UPDATE_PERIOD TCR1 counts, the initial commutation state will be present on the COMM channel pins.
10. Initialize the HALLD function.
11. The HALLD function supplies link requests along with a new STATE_NO when a commutation update is required.

6.2 Sensorless Link Update Mode (HSQ[1:0] = %10)

Assumes use in conjunction with the DUC TPU function.

1. Disable all COMM channels by clearing their two channel priority bits.
2. Select the COMM function on the selected master channel by writing the assigned COMM function number to the appropriate channel function select bits.
3. Select the desired mode of operation by writing the two host sequence bits (HSQ[0:1]) of the master COMM channel.
4. Write NO_OF_PINS, COUNTER_ADDR and UPDATE_PERIOD.
5. Write the STATE_0 to STATE_N parameter table in the slave channel(s) parameter RAM.
6. Initialize the DUC function.
7. Write a value into STATE_NO corresponding to the desired initial commutation state. Write the NO_OF_STATES parameter at the same time.
8. Issue an HSR %10 to the master channel to output the above state.
9. Write a non-zero value to the master channel priority bits to enable and prioritize service.
10. Wait for the HSR bits to be cleared by the TPU or the interrupt status bit of the master channel to be set. This indicates the end of service. Within UPDATE_PERIOD TCR1 counts, the initial commutation state will be present on the COMM channel pins.
11. Write the initial UPPER and LOWER parameters calculated from the DUC position count value obtained after stop 10.
12. Initialize OFFSET.
13. The COMM function now runs automatically as a slave to the DUC function.

6.3 Sensorless Match Update Mode (HSQ[1:0] = %00 or %01)

Assumes use in conjunction with QDEC or FQD TPU functions.

1. Disable all COMM channels by clearing their two channel priority bits.
2. Select the COMM function on the selected master channel by writing the assigned COMM function number to the appropriate channel function select bits.
3. Select the desired mode of operation by writing the two host sequence bits (HSQ[0:1]) of the master COMM channel.
4. Write NO_OF_PINS, COUNTER_ADDR and UPDATE_PERIOD.
5. Write the STATE_0 to STATE_N parameter table in the slave channel(s) parameter RAM.
6. Initialize the QDEC or FQD function.
7. Write a value into STATE_NO corresponding to the desired initial commutation state. Write the NO_OF_STATES parameter at the same time.
8. Issue an HSR %10 to the master channel to output the above state.
9. Write a non-zero value to the master channel priority bits to enable and prioritize service.
10. Wait for the HSR bits to be cleared by the TPU or the interrupt status bit of the master channel to be set. This indicates the end of service. Within UPDATE_PERIOD TCR1 counts later, the initial commutation state will be present on the COMM channel pins.
11. Write the initial UPPER and LOWER parameters calculated from the QDEC or FQD position count value obtained after step 10.
12. Initialize OFFSET.
13. Issue an HSR %11 to the master channel to start the periodic state update matches.
14. The COMM function now runs automatically, accessing the QDEC or FQD channel as required.

7 Performance and Use of Function

Like all TPU functions, the performance limit of the COMM function in a given application depends upon the service time (latency) of other active TPU channels. This is due to the way the scheduler operates. 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, information in the state timing table below, and figures in the state timing tables of other active functions.

Since COMM is always used in conjunction with at least two other TPU functions — an input function and a PWM function — the entire system must be analyzed to assess performance. In a typical system using a shaft encoder, the service demands of the quadrature function are much greater than that of the COMM function, and so determine the maximum attainable motor rpm. Typically, the higher the resolution of the encoder, the lower the maximum supportable motor rpm. In a three-phase system with a 2000 count encoder, typical maximum motor speed is approximately 11,000 rpm. If the COMM function is used with the HALLD function and no quadrature decoding takes place, or quadrature decoding is performed off-chip, maximum motor rpm is much higher (> 100,000 rpm).

Two examples of system performance calculation are provided later in this note.

Table 1 Commutation Function — State Timing

State Number and Name	Max. CPU Clock Cycles	RAM Accesses by TPU
S1 FORCE_COMM	24 + (14 ∗ NO_OF_PINS)	5
S2 UPDATE_COMM	64 + (14 ∗ NO_OF_PINS)	16
S3 MATCH_UPDATE_COMM	62 + (14 ∗ NO_OF_PINS)	16

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

7.1 Commutation Accuracy

The TPU is essentially a software (microcode) driven system. This means that there is always a delay (variable) between a change in input conditions and a corresponding change in output conditions. This delay is largely dependent on the service times of the functions running on the TPU. In the case of the COMM function this delay affects the accuracy of motor commutation, i.e. how soon the commutation state changes in response to a valid change in feedback conditions. Sensored and sensorless modes are considered separately below. In both cases it should be noted that, unless parameters are modified on the fly by the CPU, the delay affecting commutation is relatively constant in the time domain, so that the percentage accuracy in angular terms worsens as motor speed increases.

7.1.1 Sensored Mode

In this configuration (see Figure 6), the following sequence of events leads to a change in commutation.

1. A transition on one of the sensor lines occurs, resulting in a service request for the HALLD function.
2. HALLD is serviced, a new state number is written to the COMM master channel and a link request is issued.
3. The COMM function is serviced as a result of the link. The new PIN CONFIG parameter containing the commutation pin states is obtained from the state table and a match is scheduled on all the COMM channels for:

$$\text{Current time} + \text{UPDATE\_PERIOD}$$

where current time is the time at the start of the COMM function service (see below).

4. When the above match occurs, the new commutation state appears on the COMM channel pins.

UPDATE_PERIOD has a minimum allowable value that guarantees that all COMM function pins transition simultaneously when the state changes. This is due to the elapsed time between the TPU reading the current time at the beginning of COMM service and setting up the match on the last slave channel at the end of the service. UPDATE_PERIOD should always be equal to or higher than this minimum which is defined as:

$$\text{UPDATE\_PERIODmin (CPU clocks)} = 64 + 14 * \text{NO\_OF\_PINS}.$$

Providing that UPDATE_PERIOD meets these criteria, the worst case delay (TDmax) from a change in sensor states to a change in commutation state can be calculated as follows:

$$\text{TDmax} = (\text{Worst case latency of HALLD}) + (\text{Worst case HALLD service time}) +$$

$$(\text{Worst case latency for COMM}) + (14 \text{ CPU clocks for COMM time slot transition [TST]}) + \text{UPDATE\_PERIOD}.$$

Refer to the TPU reference manual and to the state timing of all functions running on the TPU for information on how to complete this calculation.

TDmax is the absolute worst case — the average delay is much smaller. This means that commutation accuracy varies in real time depending on the current loading of the TPU when a sensor transition occurs. This variation shows up as jitter on the COMM function outputs.

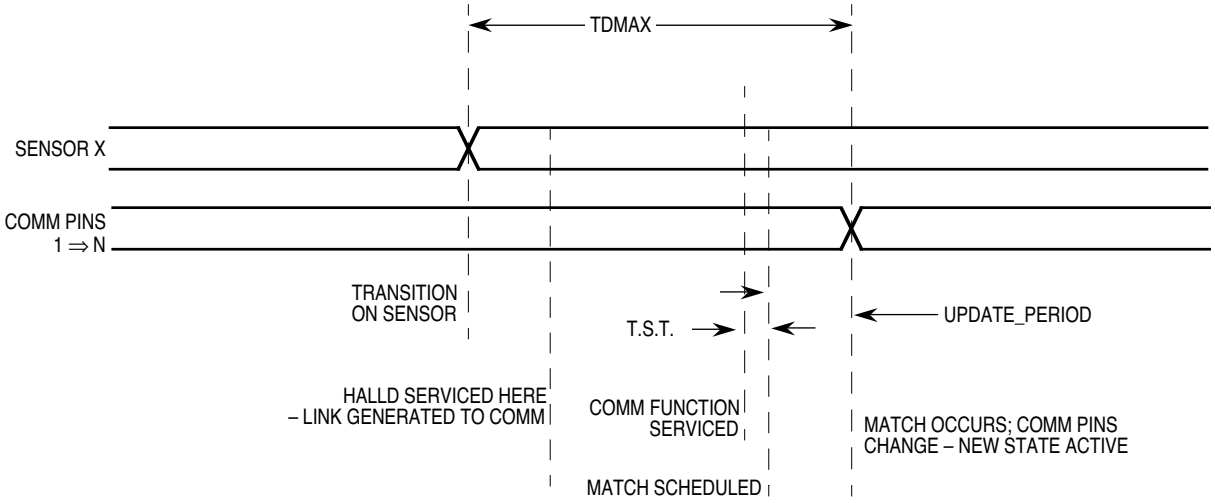

Figure 6 Sensored Configuration

7.1.2 Sensorless Mode

In this configuration (See **Figure 7**), the following sequence of events leads to a change in commutation.

1. A transition that should result in a commutation change occurs on one of the encoder sensor lines, resulting in a service request to the FQD or QDEC function.
2. The quadrature decode function services the transition and updates the position counter.
3. On the next periodic service of COMM after the position counter has been updated, new commutation pin states are obtained from the table and another periodic match is scheduled on all the COMM channels for time:

   last periodic match time + UPDATE_PERIOD.
4. When the above match occurs, the new commutation state appears on the COMM channel pins.

Here the worst case occurs when an encoder transition takes place just as a periodic service of the COMM function is started. The position counter is not updated until after the COMM function is serviced, and so is not recognized by COMM until the next periodic service — the new state does not become active until the update match after that. Worst case delay (TDmax) from a change in encoder state to a change in commutation state can be calculated as follows:

$$TDmax = 2 * UPDATE\_PERIOD$$

The average TD is approximately 1.5 times UPDATE_PERIOD, with a minimum of UPDATE_PERIOD. This variation in TD means that commutation accuracy varies in real time depending on how far through an UPDATE_PERIOD the system is when the sensor transition occurs. This variation shows up as jitter on the COMM function outputs.

The above formula assumes that UPDATE_PERIOD is longer than the sum of worst case latency and worst case service time for the COMM function. Superior performance is therefore obtained by calculating these parameters and programming UPDATE_PERIOD to be just greater than their sum. See **8 Function Examples** for more information.

If this delay is unacceptable, the OFFSET parameter can be used to make a partial correction. By using OFFSET to advance the commutation switching angles by an amount equivalent to 1.5 times

UPDATE_PERIOD TCR1 clocks at the current motor speed, switching delay can be reduced from the range –0 to +(2 ∗ UPDATE_PERIOD) to a range of approximately ± UPDATE_PERIOD / 2. To optimize this technique, OFFSET should be changed by the CPU as the motor speed varies.

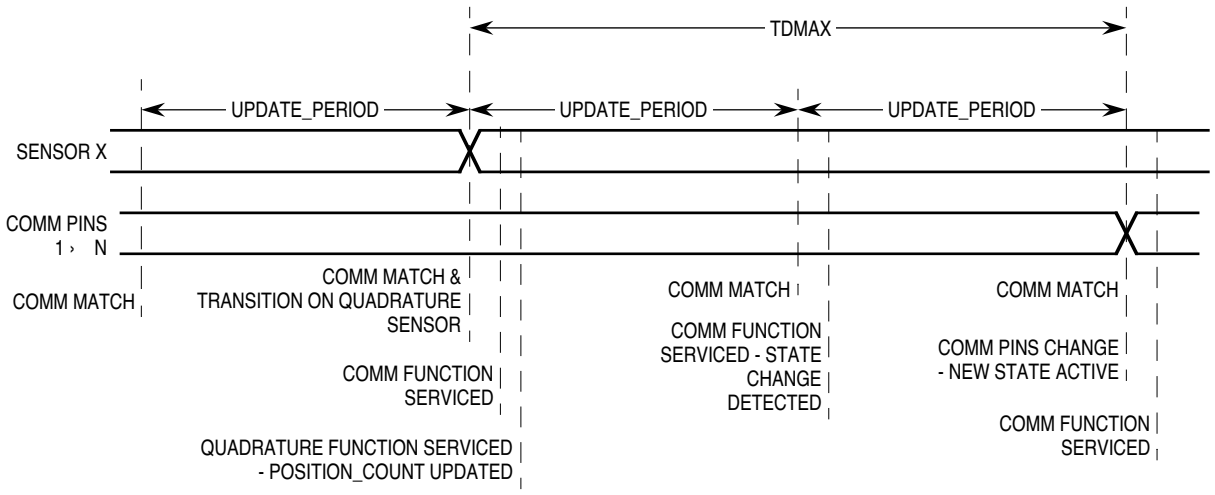

Figure 7 Sensorless Configuration

7.2 Motor Start-Up in Sensorless Mode

When commutating from an encoder, the motor must first be brought into a known rotor alignment. The COMM function allows the CPU to align the motor by forcing a sequence of commutation states, using a write to STATE_NO followed by issuing an HSR type%10. The precise sequence of states depends upon motor topology, but the sequence ends with the motor aligned in a known commutation state. If there is any load torque on the motor, it will not be perfectly center aligned within the state — this is the normal reason for providing an index mark (usually on the encoder) that produces a pulse when the rotor is perfectly aligned in the center of a particular commutation state. The mark allows the motor to be correctly aligned by slowly rotating the shaft while monitoring the index signal, either directly by the CPU or by using the NITC TPU function. When the index pulse is detected, system initialization can be completed and the COMM function will be correctly configured with the motor in the center of a known state. The following is a suggested method (others exist and may be more convenient) of completing system initialization when the index pulse is detected:

The CPU should initialize UPPER and LOWER to correspond to the boundaries of the aligned state. OFFSET should be cleared. UPPER and LOWER are derived as follows:

$$UPPER = POSITION\_COUNT + STATE\_LENGTH / 2$$

$$LOWER = POSITION\_COUNT – STATE\_LENGTH / 2$$

Where POSITION_COUNT is obtained from FQD, QDEC, or DUC function parameter RAM (POSITION_COUNT can be latched by the NITC function that detects the index pulse).

When using POSITION_COUNT for match update, issue an HSR type %11 to the master COMM channel to start the periodic matches. When using POSITION_COUNT for link update, the DUC function should be enabled to issue periodic link requests to the COMM master channel.

To actually start the motor moving in either direction, the OFFSET parameter should be written by the CPU to a value corresponding to the angular offset that produces maximum motor torque (for example, ± 1.5 times STATE_LENGTH for a three-phase motor). The sign of OFFSET determines the motor direction.

This offset causes the commutation state to change on the next periodic update of the COMM function, which in turn causes the motor to move, provided sufficient power is being applied via the PWM function. As the motor turns, the POSITION_COUNT value is updated by the TPU input function and monitored by the periodic servicing of the COMM function. As the motor aligns to the new commutation state, the OFFSET value again causes the state number to advance, thus precipitating another state change and further motor revolution. This continuous process, now without CPU intervention, keeps the motor running in the required direction.

7.3 Motor Start-Up in Sensored Mode

Motor start-up is much simpler in sensored mode than in sensorless mode. The sensors on the motor give an immediate indication of the rotor alignment. The HALLD function decodes the sensor inputs along with a desired direction input from the CPU (in HALLD parameter RAM), writes a current state number into the COMM master channel parameter RAM, then issues a link to the COMM master channel. If the state table in slave channel parameter RAM has been correctly initialized, the COMM function will output the pin configuration that causes the motor to move in the desired direction when power is applied to the phase drivers via the PWM function. To avoid generation of incorrect states, the HALLD function must be initialized before the COMM function.

7.4 Using Comm with the HALLD Function

There are two reasons care must be taken when programming the COMM state table for use with the HALLD function:

1. The HALLD function performs a straight binary demultiplex of the sensor signals and a CPU supplied direction input. The state table is not used sequentially as the motor rotates — care must be taken to enter the state parameters in the correct order.
2. The HALLD function does not screen out invalid states. In a three-phase configuration, the three sensor signals, along with the CPU supplied direction input, can decode any number between 0 and 15, but only twelve of these are valid. The other four values could result from noise on the sensor lines. The user must provide a 16-entry state table in the COMM function, with the four invalid states programmed to take an appropriate action (usually to disable all phase drivers). The COMM master channel must be channel 13, so that there is sufficient contiguous parameter RAM to hold the table. Similarly, a four phase motor requires a table size of eight entries. In this case either channel 13 or 14 can be the COMM master channel. **Figure 8** shows a suitable table configuration for a three-phase motor. Commutation waveform diagrams for both directions of motion can be deduced from the table. In the table contents, N is equal to the bit value that causes the phase driver to be turned off.

HALL C	HALL B	HALL A	DIRECTION	STATE_NO	COMM State Table
0	0	0	0	0	XXXXXXXX XX101100
0	0	0	1	1	XXXXXXXX XX011010
0	0	1	0	2	XXXXXXXX XX110100
0	0	1	1	3	XXXXXXXX XX011001
0	1	0	0	4	XXXXXXXX XXNNNNNN
0	1	0	1	5	XXXXXXXX XXNNNNNN
0	1	1	0	6	XXXXXXXX XX110010
0	1	1	1	7	XXXXXXXX XX101001
1	0	0	0	8	XXXXXXXX XX101001
1	0	0	1	9	XXXXXXXX XX110010
1	0	1	0	10	XXXXXXXX XXNNNNNN
1	0	1	1	11	XXXXXXXX XXNNNNNN
1	1	0	0	12	XXXXXXXX XX011001
1	1	0	1	13	XXXXXXXX XX110100
1	1	1	0	14	XXXXXXXX XX011010
1	1	1	1	15	XXXXXXXX XX101100

Figure 8 State Table for Three-Phase Motor Using Sensored Mode and HALLD

7.5 Using Offset in Sensorless Mode

The OFFSET parameter has three basic uses:

1. Enabling motion in a defined direction as described in **7.2 Motor Start-Up in Sensorless Mode**.
2. Compensating for commutation delays due to service time as described in **7.1 Commutation Accuracy**.
3. To advance and retard the commutation firing angles while the motor is running.

This last use allows the commutation states to be advanced by a variable amount at high motor speeds for torque maintenance, and to be retarded to assist braking of the motor.

OFFSET can be used in all three ways in a single application by making it one of the primary outputs of the CPU control algorithm, so that it is continually updated along with the duty cycle of the PWM function.

7.6 Interrupts

A CPU interrupt request is generated by the COMM master channel on completion of every service of the function. These interrupts can be enabled or disabled under CPU control via the channel interrupt enable register (CIER). When interrupts are disabled, the bit in the channel interrupt status register (CISR) corresponding to the COMM master channel is set on completion of every service of the function, allowing it to be polled by software.

The interrupt or interrupt status bit can be used to monitor completion of service after a host service request has been issued by the CPU, or to synchronize other events in the system to the regular servicing of the COMM function.

8 Function Examples

Since the configuration of the COMM function must be viewed as part of an overall system, and many parameters are dependent on the motor specification, no specific programming examples are given in this note. However, guidelines on the calculation of possible system performance are required and two examples follow. Refer to the documentation for each of the functions mentioned in the examples for timing figures and more detail. Refer also to the latency section of the TPU reference manual. The performance figures derived below are representative only — a detailed analysis is recommended for any proposed implementation.

8.1 Example A —Sensorless Mode

8.1.1 Description

A three-phase brushless motor system is to be designed around the TPU. The commutation scheme is sensorless, uses a 1000 count per revolution encoder, and employs the quadrature decoding capabilities of the TPU fast quadrature decode (FQD) function. The NITC function is used to capture the index pulse from the encoder. Two channels are used by the multichannel PWM (MCPWM) TPU function, configured in edge-aligned mode to drive the motor with a 20-kHz PWM. Six channels are used by the COMM function, to produce commutation drive signals that gate the PWM onto the motor phases. The MCU is clocked at 20.97 MHz. Calculate the maximum motor rpm that can be supported by the TPU in such a configuration and the accuracy of the commutation. A schematic of the system is shown in **Figure 9**.

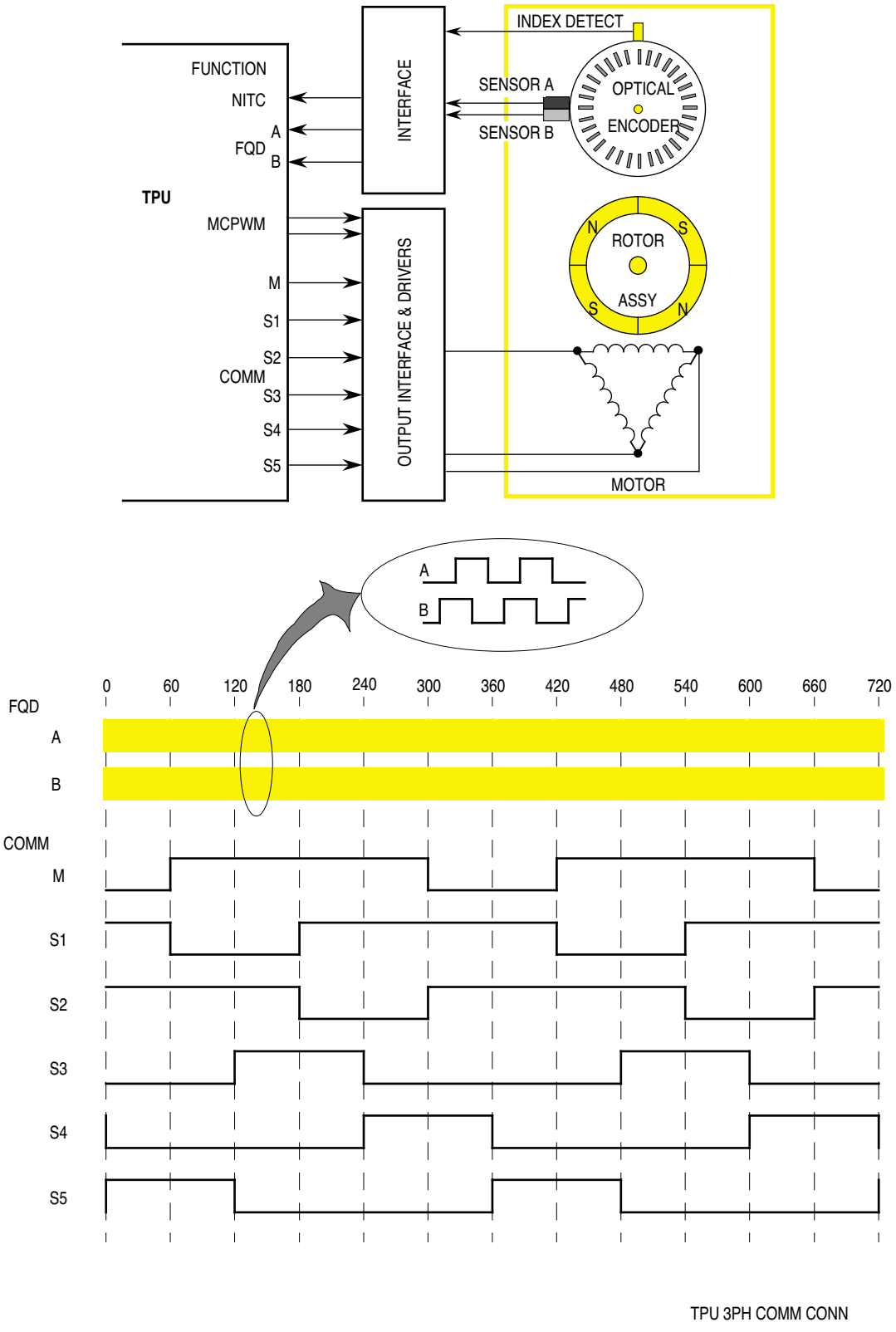

Figure 9 Example Brushless System

8.1.2 Analysis

Assuming that at high motor speeds the period of the quadrature signals is much smaller than the period of either the PWM function or the update period of COMM, it is most efficient to assign high priority to the FQD and NITC signals and medium priority to the other functions. This ensures that the quadrature decode function is given the majority of the available TPU processing time slots. The NITC function must also be assigned high priority to correctly capture the position count value when the index pulse occurs, but it need only be active during motor start-up when the slow speed of the motor ensures that there will be no TPU loading problems. Since it is more critical for the MCPWM function to be serviced on time than the COMM function, MCPWM is assigned to lower channel numbers than COMM. This ensures that although both functions have the same priority, MCPWM will be serviced first if both functions request at the same time. A suitable channel assignment for the application follows.

CHANNEL	FUNCTION	PRIORITY
0	NITC	H
1	FQD	H
2	FQD	H
3	MCPWM master	M
4	MCPWM slave	M
5	COMM master	M
6	COMM slave1	Disabled
7	COMM slave2	Disabled
8	COMM slave3	Disabled
9	COMM slave4	Disabled
10	COMM slave5	Disabled

There are six electrical states in the three-phase configuration and two electrical rotations for one physical rotation of the rotor. This implies that each commutation state is 1000/12 = 83.333 encoder counts in length. The COMM function supports this fractional state length via the programmable state table; since it is impossible to program a partial encoder count into the STATE_LENGTH field, the states can be programmed to different lengths, for example:

STATE_LENGTH0	83
STATE_LENGTH1	84
STATE_LENGTH2	83
STATE_LENGTH3	83
STATE_LENGTH4	84
STATE_LENGTH5	83
	500 ∗ 2 = 1000

Thus 1000 encoder counts now correspond to exactly two electrical revolutions.

If a very high resolution encoder is used, so that STATE_LENGTH > 256, 12 state parameters are required, with pairs sharing the same PIN_CONFIG field. This makes it necessary to move the COMM master channel to channel 13 or channel 14 to provide sufficient contiguous parameter RAM. The slave channels would be 14 to 2 or 15 to 3 respectively. NO_OF_STATES would be programmed to 12.

8.1.3 Performance Calculation

8.1.3.1 Assumptions

1. No TPU functions are running on channels 11 to 15.
2. The NITC function is disabled after motor start up and thus does not figure in high speed analysis.
3. At high motor speed, FQD is used in the fast mode.
4. The CPU does not issue TPU host service requests while the motor is running at high speed.
5. The RAM collision rate (RCR) between the TPU and CPU is very small (see reference manual). RCR is not included in the calculations. If significant, RCR can have a major impact on TPU performance. RCR can be reduced by avoiding situations such as tight software loops that poll the TPU parameter RAM. If RCR is considered to be significant, the calculations below should be repeated using a representative RCR and the procedures described in the TPU reference manual.

8.1.3.2 Performance Data:

Worst case service time of FQD in fast mode = 26 CPU clocks
Worst case service time of FQD in normal mode = 46 CPU clocks
Worst case service time of NITC (single shot, no links) = 38 CPU clocks
Worst case service time of MCPWM master = 28 CPU clocks
Worst case service time of MCPWM slave = 26 CPU clocks
Worst case service time of COMM master = 74 + (14 ∗ NO_OF_PINS) = 158 CPU clocks
The above times include a time slot transition time (TST) of 10 clocks.

Calculation 1: Maximum Motor RPM

The first intuitive assumption is that the servicing of the FQD function is the most critical aspect of the system — if an FQD edge is not serviced before the next signal edge occurs, then that edge will be lost and the position count corrupted. Scheduler time slot assignment produces the following scheme:

HMHLHMHHMHLHMH...

Since there are no low priority channels, the low time slots are given to the highest priority requesting channel at the time. Similarly, if no high priority channel is requesting at the start of a high priority time slot then the slot is assigned to the next highest priority channel that is requesting.

The worst case for service of an FQD signal edge occurs when the transition occurs just as a high priority time slot is given away to a medium channel. If the time slot after that is a “real” medium priority, the FQD transition must wait while two medium channels are serviced before it is serviced in the next high or low priority time slot. The minimum allowable time between FQD transitions is therefore the length of time taken to service these two medium channels and the time taken to service the FQD channel.

For worst case analysis, the two longest medium service times are used. If it is assumed that the second medium channel is the last medium channel requesting at the time, then four additional CPU clocks must be added to the calculation to allow for the resetting of the service grant latch (see reference manual). Similarly, since FQD service must be complete before the next FQD transition, the high priority service grant latch must be cleared after each FQD service, requiring the addition of four more clocks.

The two worst case medium services are the COMM function and the MCPWM master channel. The minimum allowable time between FQD transitions is therefore:

158 + 28 + 26 + 8 = 220 CPU clocks

At 20.97 MHz, this is equivalent to a minimum period of 220 ∗ 47.69 ns = 10.49 µs.

In FQD fast mode there are four encoder counts for every serviced transition.

Minimum count period = 10.49/4 = 2.623 µs

At 1000 counts per revolution, this equates to a maximum motor speed of 22,875 rpm.

Since the critical factor is FQD servicing, this maximum motor rpm is proportional (up to a point) to the resolution of the encoder. For example, if the resolution is doubled to 2000 counts per revolution, then maximum motor speed would be reduced to approximately 11,400 rpm.

Calculation 2: Maximum Motor RPM With FQD in Normal Mode

Calculation 1 assumes that FQD is running in fast mode at high motor speeds. However, the function must run in normal mode at lower speeds to ensure correct direction decoding and full accuracy. It is therefore necessary to calculate the maximum motor speed that can be sustained in normal mode so that the CPU can switch modes at the most efficient speed.

In normal mode a transition on one of the FQD channels must be serviced before the next transition on the other channel, to ensure correct direction decoding. The minimum time between transitions is therefore the maximum latency of an FQD service plus the service time of FQD in normal mode.

From calculation 1:

Minimum time between transitions = 158 + 28 + FQD service time + 8

= 158 + 28 + 46 + 8 = 240 CPU clocks

At 20.97 MHz, this is equivalent to a minimum period of 240 ∗ 47.69 ns = 11.45 µs.

In normal mode this equates to the period of one count.

At 1000 counts per revolution, this equates to a maximum motor speed of 5240 rpm.

Calculation 3: Maximum Commutation Angle Error

Having established the maximum motor speed, it is also useful to calculate the minimum sustainable COMM function UPDATE_PERIOD, as a minimum value will result in the most accurate commutation. Once this value is established, maximum switching angle error can be determined for a given motor rpm.

The minimum value for UPDATE_PERIOD is defined by the sum of worst case latency for COMM plus worst case service time for COMM. Since in this example UPDATE_PERIOD is constant from motor start-up, the service time of the NITC channel must also be taken into account. The worst case latency for COMM therefore occurs when all other functions are requesting service as well, and the scheduler is at the following point in its sequence:

HHMHLHM...

The first high priority time slot is given to the FQD function: 46 CPU clocks

The second high priority time slot is given to the NITC function: 38 + 4 CPU clocks

The first medium time slot is given to the MCPWM master channel: 28 CPU clocks

The third high priority time slot is given to the FQD function if another transition has occurred since the service of the previous one (latency of last FQD transition is assumed to be maximum therefore another transition could occur immediately after it was serviced): 46 + 4 CPU clocks.

The next time slot is a low priority, but since FQD has just been serviced and only one NITC transition occurs per revolution, this time slot is given away to the medium priority MCPWM slave: 26 CPU clocks.

The next time slot is another high priority, but since a maximum of only 146 CPU clocks can have elapsed since the last FQD transition, and since the minimum time between FQD transitions in normal mode has been defined by calculation 2 to be 240 CPU clocks, there is no pending high priority channel and the time slot is given to the medium priority COMM channel.

The worst case latency for COMM is therefore = 46 + 42 + 28 + 50 + 26 = 192 CPU clocks

The worst case service time of COMM is 158 + 4 (as this is the last medium channel) = 162 CPU clocks

The minimum update period of COMM is therefore 192 + 162 = 354 CPU clocks.

With TCR1 configured for divide by 4 operation, this results in a minimum UPDATE_PERIOD of 89.

As defined under **7.1 Commutation Accuracy** the worst case switching delay is equivalent to

$$-0, +(2 * \text{UPDATE\_PERIOD}).$$

This is equal to 712 clocks or 33.96 μs.

At 20,000 rpm this is equivalent to an angle error of 33.96/8.33 = 4.08 degrees.

Angle error is proportional to motor speed, so that at 2000 rpm it equals 0.408 degrees, and so on.

Notes

1. Angle error can be improved considerably by varying UPDATE_PERIOD with motor speed and/or using OFFSET to move the center point of the error (See **7.1 Commutation Accuracy**). At high motor rpm, NITC should not be active and FQD service time in fast mode is shorter, therefore possible minimum values for UPDATE_PERIOD are smaller and there is a smaller angle error. Applying these two methods could reduce the worst case error to a quarter of the value calculated above.
2. If the maximum obtainable rpm of the motor calculated above is too low, it can be increased by using only three channels for the COMM function and adding external decode logic. The three COMM channels are configured to output the required state number and the external logic decodes the state number into commutation drive signals. In this case, the worst case service time of the COMM function would be 116 CPU clocks, resulting in a maximum motor rpm of approximately 28,200 rpm.

8.2 Example B —Sensored Mode

8.2.1 Description

A three-phase brushless motor system is to be designed around the TPU. The commutation scheme employed is sensored, using three Hall effect sensors decoded by the TPU Hall effect decode (FQD) function. Two channels are used by the multichannel PWM (MCPWM) TPU function, configured in edge-aligned mode to drive the motor with a 20-kHz PWM. Six channels are used by the COMM function to directly produce commutation drive signals that gate the PWM onto the motor phases. The MCU is clocked at 16.78 MHz. A schematic of the system is shown in **Figure 10**. Calculate the maximum motor rpm that can be supported by the TPU in such a configuration.

8.2.2 Analysis

In sensored or link mode, each transition and service of the HALLD function causes a service of the COMM master channel. Since the time taken to service these two functions directly affects commutation accuracy, both functions must be assigned high priority in order to minimize this time. The MCPWM function can be assigned a medium priority. Since a 16-entry state table is required in sensored mode, the COMM master channel must be channel 13 or channel 14. A suitable channel assignment for the application follows.

CHANNEL	FUNCTION	PRIORITY
3	HALLD	H
4	HALLD	H
5	HALLD	H
6	MCPWM master	M
7	MCPWM slave	M
13	COMM master	H
14	COMM slave1	Disabled
15	COMM slave2	Disabled
0	COMM slave3	Disabled
1	COMM slave4	Disabled
2	COMM slave5	Disabled

There are six electrical states in the three-phase configuration and two electrical rotations for one physical rotation of the rotor. This implies that there are 12 sensor transitions in one rotation of the rotor.

8.2.3 Performance Calculation

8.2.3.1 Assumptions

1. No TPU functions are running on channels 8 to 12.
2. The RAM collision rate (RCR) between the TPU and CPU is very small (see reference manual). RCR is not included in the calculations. The CPU must spend a significant percentage of time accessing parameter RAM for RCR to be an issue. If RCR is considered to be significant, the calculations below should be repeated using a representative RCR and the procedures described in the TPU reference manual.

8.2.3.2 Performance Data:

Worst case service time of MCPWM master = 28 CPU clocks
Worst case service time of MCPWM slave = 26 CPU clocks
Worst case service time of HALLD with three sensors = 80 CPU clocks
Worst case service time of COMM master = 34 + (14 ∗ NO_OF_PINS) = 118 CPU clocks
The above times include a time slot transition time (TST) of 10 clocks.

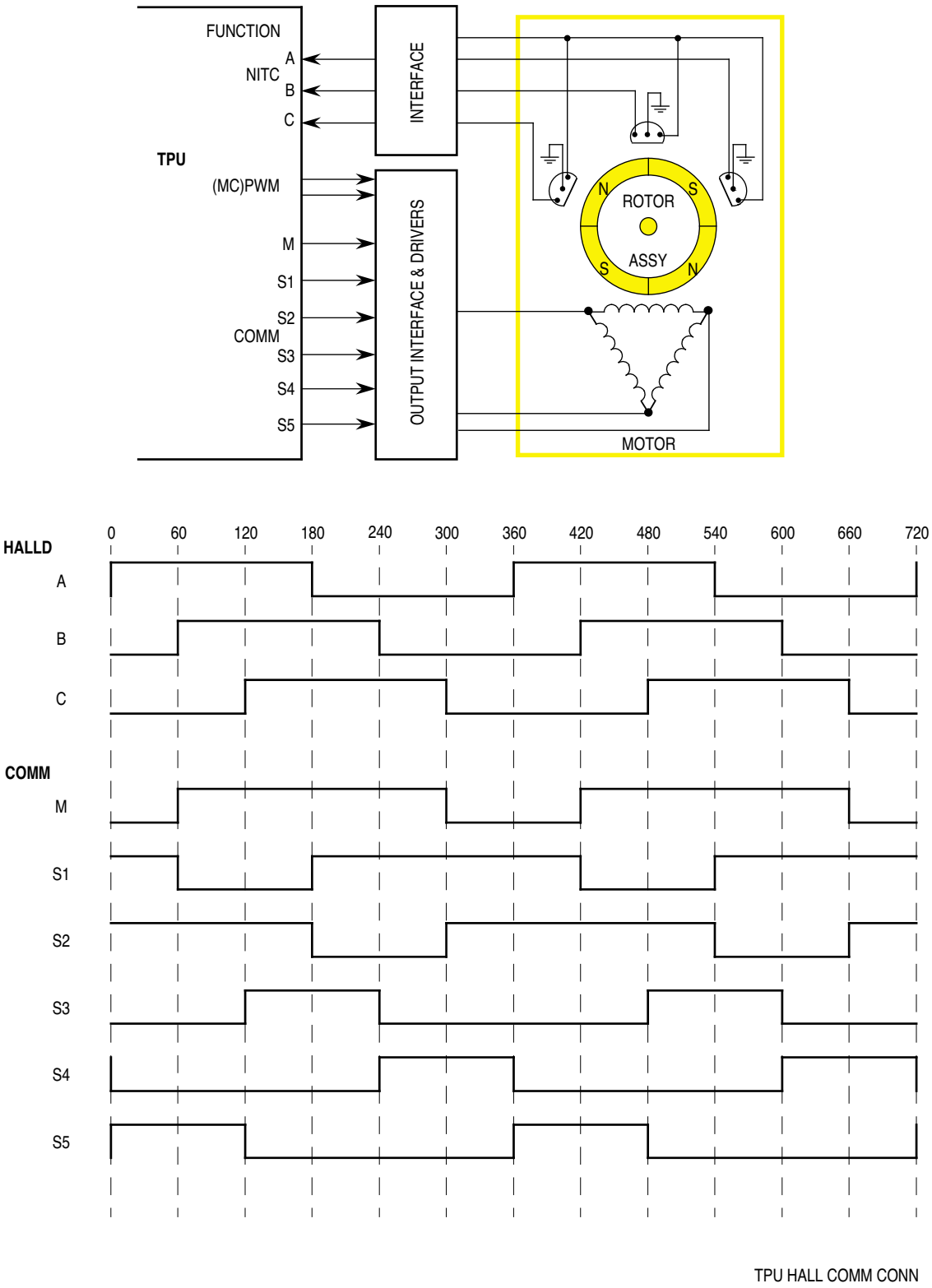

Figure 10 Example Hall Effect System

8.2.3.3 Calculation: Maximum Motor RPM

To calculate the maximum attainable motor rpm, it is necessary to calculate the minimum allowable period between HALLD sensor transitions that can result in all possible pending services being completed on time. The time slot assignment of the scheduler produces the following scheme:

HMHLHMHHMHLHMH...

Since there are no low priority channels, the low time slots are given to the highest priority channel that is requesting service. Similarly, if no high priority channel is requesting service at the start of a high priority time slot, the slot is assigned to the next highest priority channel that is requesting service.

Worst case for the service of a HALLD signal edge occurs when a transition occurs just as a high priority time slot is given away to a medium priority channel. If the time slot after that is a “real” medium priority, the HALLD transition must wait while two medium priority channels are serviced before it is serviced, in the next high or low priority time slot. The HALLD function generates a link request to the COMM master channel after it is serviced, and the COMM master must also be serviced before the next HALLD transition.

The minimum allowable time between HALLD transitions is therefore the length of time taken to service the two medium priority channels, plus the time taken to service COMM and HALLD. There are also two resets of the service grant latch during this time, because all medium priority channels and all high priority channels are serviced during this time — an extra eight CPU clocks must be added to the calculation (see reference manual). The two medium priority services are the master and slave MCPWM channels.

The minimum allowable time between HALLD transition is therefore:

80 + 118 + 28 + 26 + 8 = 260 CPU clocks

At 16.78 MHz, this is equivalent to a minimum period of 260 ∗ 59.6 ns = 15.5 µs.

This equates to a maximum motor speed of 60 ∗ 1/(15.5 µs ∗ 12) = 322,580 rpm.

This is the maximum possible speed, but it results in a lag of a full commutation state, because the state corresponding to a particular sensor configuration is just established as the next sensor transition occurs. For this reason, maximum practical motor speed is less than this figure.

9 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 Freescale 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 commutation function consists of three states, which operate as described below. For clarity, reference is made to internal channel flag0 in the following descriptions. This internal TPU control bit is not available to the user.

9.1 State 1: FORCE_COMM

This state is entered as a result of an HSR type %10 or a link request with internal channel flag0 = 0 (sensored mode after initialization). This state forces the pin configuration obtained from the state parameter corresponding to the current value of STATE_NO to be output on the COMM function pins as follows:

```
TEMP = TCR1
Disable servicing of match events
If host sequence bit 0 (HSQ0) is set
    Clear channel flag0
Else
    Set channel flag0
Get STATE_N from slave channel parameter RAM using STATE_NO as table address.
Set up COMM channels to output the state defined by the PIN CONFIG field of STATE_N when
    the next match on each channel occurs
Schedule a match on each COMM channel for time TEMP + UPDATE_PERIOD
Generate a CPU interrupt request from the master COMM channel
End
```

When the scheduled match occurs, the new pin configuration will appear on the appropriate COMM channel pins.

9.1.1 State 2: UPDATE_COMM

This state is entered as a result of an HSR type %11 or a link request with internal channel flag0 = 1 (sensorless link mode after initialization). This state tests the remote position counter and OFFSET against the current state boundaries and updates the commutation state if either boundary has been crossed. The state operates as follows:

```
TEMP = TCR1
Goto TEST_STATE
```

9.1.2 State 3: MATCH_UPDATE_COMM

This state is entered as a result of a match service request in sensorless match update mode. This state tests the remote position counter and OFFSET against the current state boundaries and updates the commutation state if either boundary has been crossed. The state operates as follows:

```
TEMP = last match time
TEST_STATE:
Get POSITION_COUNT via COUNTER_ADDR
TEMP2 = POSITION_COUNT + OFFSET
If TEMP2 < LOWER then
        STATE_NO = STATE_NO – 1
        If STATE_NO negative then STATE_NO = NO_OF_STATES
            Get STATE_N from slave channel parameter RAM using STATE_NO as table address
            UPPER = LOWER
            LOWER = LOWER – STATE_LENGTH
Else if TEMP2 ≥ UPPER then
        STATE_NO = STATE_NO + 1
        If STATE_NO > NO_OF_STATES then STATE_NO = 0
            Get STATE_N from slave channel parameter RAM using STATE_NO as table address
            LOWER = UPPER
            UPPER = UPPER + STATE_LENGTH
Endif
If host sequence bit 1 is clear (match mode) then
        Enable servicing of match events
Else
        Disable servicing of match events (link mode)
        If host sequence bit 0 (HSQ0) is set
            Clear channel flag0
        Else
            set channel flag0
Endif
Get STATE_N from slave channel parameter RAM using STATE_NO as table address
Set up COMM channels to output the state defined by the PIN CONFIG field of STATE_N when
        the next match on each channel occurs
Schedule a match on each COMM channel for time TEMP + UPDATE_PERIOD
Generate a CPU interrupt request from the master COMM channel
End
```

When the scheduled match occurs, the new pin configuration will appear on the appropriate COMM channel pins.

NOTES

NOTES

NOTES

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