

# PERIOD MEASUREMENT WITH ADDITIONAL TRANSITION DETECTION (PMA)

by Sharon Darley

## 1 Functional Overview

The PMA function detects additional transitions embedded in a series of input pulses by measuring each pulse period to a 23-bit resolution. It detects an additional transition when the current period is less than the previous period multiplied by a programmable ratio. It has two operating modes: count mode and bank mode. In count mode, the PMA function counts the number of additional transitions and compares it with a programmable maximum value before resetting the TCR2 counter and starting over with the next series of pulses. In bank mode, the TCR2 counter resets when an additional transition is detected and the flag BANK\_SIGNAL is set to a non-zero value.

## 2 Detailed Description

The PMA function is typically used in automotive applications for detecting a reference point on a fly-wheel with regularly spaced teeth. This reference point is in the form of an additional tooth. An additional tooth causes a shorter interval between teeth, and the PMA function is able to detect this shorter interval by measuring the current period and comparing it with the previous period. The PMA function is usually used in conjunction with the PSP function. The PSP function generates an output pulse in relation to the additional tooth detected by the PMA function.

The PMA function measures the period between regularly spaced transitions for a channel that has its input connected to the same source as the clock input to TCR2. It detects an additional transition when the current period  $< \text{PERIOD} * \text{RATIO}$ . Thus, it is able to effectively map engine-cycle position into TCR2 counts.

When the PMA function is in count mode and detects an additional transition, it increments the parameter ADDITIONAL\_COUNT. Then, it compares the new ADDITIONAL\_COUNT to MAX\_ADDITIONAL. If ADDITIONAL\_COUNT is greater than or equal to MAX\_ADDITIONAL, then the PMA function will reset TCR2 to \$FFFF, clear ADDITIONAL\_COUNT, and request an interrupt.

When the PMA function is in bank mode and detects an additional transition, the TPU reads the value BANK\_SIGNAL. If BANK\_SIGNAL is set to a non-zero value, the PMA function will reset TCR2 to \$FFFF, clear BANK\_SIGNAL, and request an interrupt. If BANK\_SIGNAL is set to zero, the PMA function will not reset TCR2, but will request an interrupt. BANK\_SIGNAL can be set by another function. The ITC function is ideal for this purpose.

ROLLOVER\_COUNT is a parameter used to calculate a 23-bit period from the 16-bit TCR count. It increments each time the TCR count equals or exceeds \$8000 during a period measurement. At the beginning of service, REF\_TIME contains the TCR1 value of the prior input capture, and the capture register contains the TCR1 value of the latest transition. On each normal transition detected (i.e., not on an additional transition), the PMA function places the elapsed time measured into PERIOD\_LOW\_WORD and PERIOD\_HIGH\_WORD, measuring a time of up to \$7EFFFF TCR1 clocks. (If the period measured is longer than \$7EFFFF, the period time is set to \$7FXXXX.)



Some transitions may be incorrect or invalid. In automotive and similar environments, noise in the system, malformed teeth on the flywheel, or incorrect adjustment of the transducer can cause extra transitions that are not a result of the flywheel teeth. To help detect these invalid transitions, the PMA supplies some noise immunity by requiring that 1) the total number of transitions just before the final additional tooth is reached does not exceed the value of TCR2\_MAX\_VALUE, and 2) the total number of transitions to be counted is equal to NUM\_OF\_TEETH. Remember that tooth count begins with zero instead of one, so the values in these two parameters will be one less than the actual number of teeth. These checks help to identify the index position. If the numbers do not match, the function assumes that the detected additional transition was invalid and responds by doing the following:

1. Indicates the error condition by setting TCR2 to the value \$80FF and the most significant byte of TCR2\_VALUE to \$80 (the value \$80FF can never cause a match detection by a channel executing PSP);
2. Interrupts the CPU on each tooth (input transition);
3. Continues to increment the lower byte of TCR2\_VALUE;
4. Continues to calculate the period;
5. Stores the time of the last tooth in REF\_TIME.

The decision of what to do for an asynchronous condition is left up to the system programmer. The programmer may then use the force mode available in the OC or PSP functions to directly control the output channels.

The system designer must ensure that the additional transition is serviced completely (both scheduled and served) before the next normal transition occurs. If service is incomplete, errors may pass undetected. To ensure complete service, there are limitations on the parameter ratio (see description) and requirements for programming the scheduler. Some basic rules are the following:

- The PMA and PSP channels must have the same priority level.
- The PMA channel must be assigned a lower channel number than the PSP channel in order to service the PMA channel before the PSP is serviced in the case of simultaneous requests.
- The priority level of the PMA channel should be high enough to ensure service completion before the next normal transition occurs.

At PMA initialization, TCR2 is initialized to \$C0FF since the value of \$C0FF cannot cause a match detection by the channel executing PSP. On each transition detected, TCR2\_VALUE is incremented, tracking TCR2, and is readable by the CPU at any time. The PMA channel is synchronized to the input whenever the additional and normal transitions occur in the expected sequence. With synchronization, TCR2 is reset to \$FFFF. TCR2 values from \$0000 to \$00FF can cause a match detection by the channels executing PSP. By setting the host sequence bit 0, the user may choose one of two operating modes: count mode or bank mode.

#### Count Mode

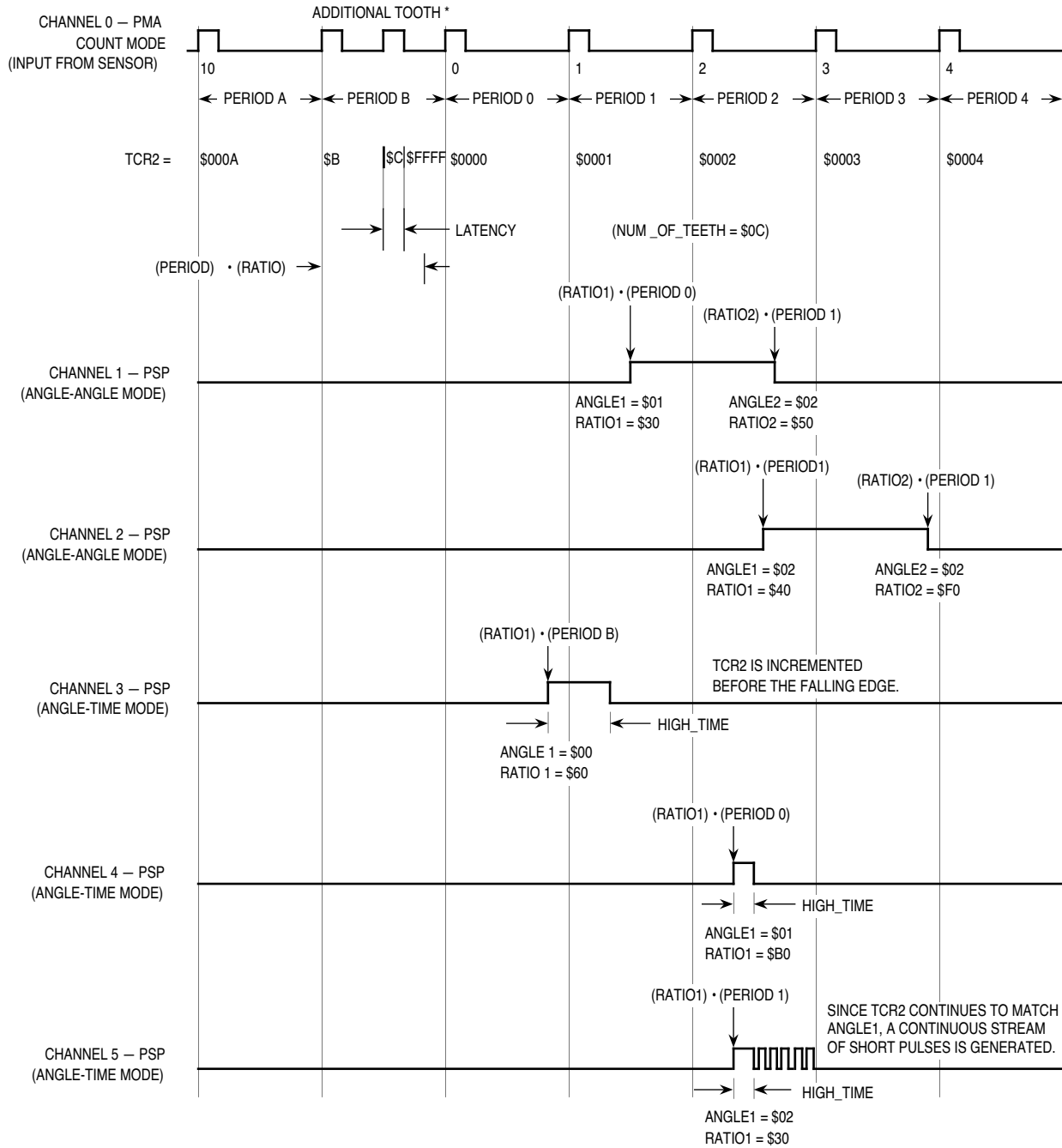
In this mode, TCR2 is set to \$FFFF after the number of additional transitions in MAX\_ADDITIONAL has been identified and counted.

#### Bank Mode

In this mode, TCR2 is set to \$FFFF if an additional transition has been identified and BANK\_SIGNAL is a non-zero value. BANK\_SIGNAL can be incremented by another function such as the ITC function.

**Figure 1** is an example of PMA used with a PSP function. The input to the PMA channel and the TCR2 clock input is a flywheel with teeth spaced every 30 degrees, with one additional tooth as a reference indicator. Detection of the additional tooth causes the function to set TCR2 to \$FFFF; the next tooth and input capture event advances TCR2 to \$0000, and each successive tooth advances TCR2. When the additional tooth is detected, TCR2 and TCR2\_VALUE contain \$0C; therefore, NUM\_OF\_TEETH must also be \$0C.

Channels 4 and 5 illustrate two ways of programming a pulse using PSP angle-angle mode. Channel 4 illustrates the preferred way since it produces only one waveform. Refer to Motorola Programming Note TPUPN14/D, *Position-Synchronized Pulse Generator (PSP)* for additional information.



\* ADDITIONAL TOOTH – ONE TOOTH EVERY 30° PLUS ONE TOOTH RESULTS IN A TOTAL OF 13 TEETH.

Figure 1 PMA and PSP Together

### 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. PMA function code size is:

$$80 \mu \text{ instructions} + 6 \text{ entries} = \mathbf{86 \text{ long words}}$$

### 4 Function Parameters

This section provides detailed descriptions of PMA function parameters stored in channel parameter RAM. **Figure 2** shows TPU parameter RAM address mapping. **Figure 3** shows the parameter RAM assignment used by the PMA function. In the diagrams, Y = M11, 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 2 TPU Channel Parameter RAM CPU Address Map**

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$YFFFW0	REF_TIME							CHANNEL_CONTROL								
\$YFFFW2	MAX_ADDITIONAL							NUM_OF_TEETH								
\$YFFFW4	BANK_SIGNAL/ADDITIONAL COUNT							ROLLOVER_COUNT								
\$YFFFW6	RATIO							TCR2_MAX_VALUE								
\$YFFFW8	PERIOD_HIGH_WORD															
\$YFFFWA	PERIOD_LOW_WORD															
\$YFFFFC	ERROR				TCR2_VALUE											

W = Channel number

Parameter Write Access:

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

**Figure 3 PMA Function Parameter RAM Assignment**

#### 4.1 CHANNEL\_CONTROL

CHANNEL\_CONTROL contains the channel latch controls and configures the PSC, PAC, and TBS fields. A channel executing this function is configured as input. The CPU must write CHANNEL\_CONTROL before initializing the function. The PSC field is “don't care” for input channels. The PAC field specifies which edge to detect. Since the TCR2 external clock input detects rising edges only, the PAC field should be configured to detect rising edges unless unusual conditions exist. The TBS field configures a channel pin as input or output and configures the time base for match/capture events. The PMA function should use TCR1 for both types of events. The following table defines the allowable data for this parameter.

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

**Table 1 PMA CHANNEL\_CONTROL Options**

TBS		PAC			PSC		Action	
8 7 6 5		4 3 2			1 0		Input	Output
		0 0 0					Do Not Detect Transition	—
		0 0 1					Detect Rising Edge	—
		0 1 0					Detect Falling Edge	—
		0 1 1					Detect Either Edge	—
		1 x x					Do Not Change PAC	—
0 0 x x							<b>Input Channel</b>	—
0 0 0 0							Capture TCR1, Match TCR1	—
1 x x x							Do Not Change TBS	—

#### 4.2 REF\_TIME

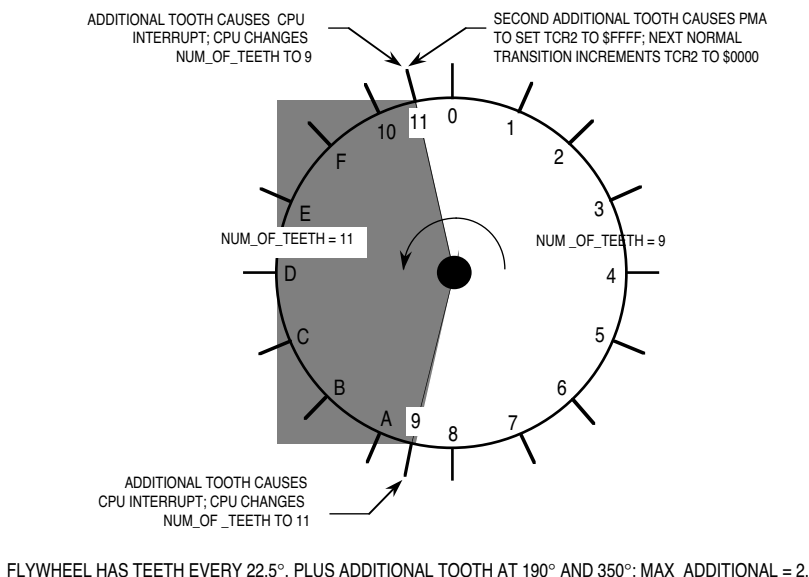
REF\_TIME is the time of the last transition captured. After CHANNEL\_CONTROL information is used during the *Init* state, the TPU writes the captured TCR1 value into REF\_TIME on each normal transition detected.

### 4.3 MAX\_ADDITIONAL

MAX\_ADDITIONAL is a byte that contains the number of additional transitions to be counted before TCR2 is set to \$FFFF. This parameter is written by the CPU before initialization and is referenced by the TPU only in count mode (host sequence bits equal 01). It cannot be set to zero and used to generate a PSP output waveform for an input pulse train with no additional teeth.

### 4.4 NUM\_OF\_TEETH

NUM\_OF\_TEETH is one less than the total count of transitions seen by TCR2 at the time any additional transition is detected. This parameter differentiates between valid and invalid transitions. For each additional transition detected, the TPU subtracts NUM\_OF\_TEETH from the current tooth count in TCR2 and continues executing the function if the result is zero. If the application has multiple additional teeth separated by different numbers of normal teeth, the host CPU must update NUM\_OF\_TEETH once for each set, so that NUM\_OF\_TEETH contains the same value as TCR2 at each additional tooth. In **Figure 4**, the flywheel is divided into two segments with different numbers of teeth in each segment, so that the CPU may easily determine which half of the flywheel has the major reference tooth. At each additional tooth detected, TCR2 is compared with NUM\_OF\_TEETH, and an interrupt is sent to the CPU to allow the software to alternately change NUM\_OF\_TEETH between \$09 and \$11.



**Figure 4 PMA NUM\_OF\_TEETH Example**

### 4.5 BANK\_SIGNAL

BANK\_SIGNAL is a byte that contains the current value of the bank signal. A non-zero value means that BANK\_SIGNAL is asserted; \$00 means that BANK\_SIGNAL is negated. This parameter is referenced only in bank mode (host sequence bits equal 00). If BANK\_SIGNAL is non-zero when a missing transition is detected, then TCR2 is set to \$FFFF and BANK\_SIGNAL is cleared. BANK\_SIGNAL is normally set by another time function, such as the ITC function, but can be set by the CPU when an outside reference determines that the next additional transition should cause TCR2 to become \$FFFF.

### 4.6 ADDITIONAL\_COUNT

In count mode, the byte ADDITIONAL\_COUNT contains the number of additional transitions detected. At initialization, this parameter is set to \$0000. Then, at each additional tooth, the PMA function compares ADDITIONAL\_COUNT with the value in MAX\_ADDITIONAL. If it is greater than or equal to MAX\_ADDITIONAL at any additional tooth, TCR2 is set to \$FFFF and ADDITIONAL\_COUNT is reset to \$0000. This parameter is used only in count mode.

#### 4.7 ROLLOVER\_COUNT

ROLLOVER\_COUNT is a parameter used as a counter that increments each time the TCR count equals or exceeds \$8000 during a period measurement. This parameter is used to calculate a 23-bit period from the 16-bit TCR count and to determine if a period error (**Figure 5**) has occurred. This parameter is reset at the beginning of each measured period.

#### 4.8 RATIO

RATIO, multiplied by the previous PERIOD, is the upper bound of time in which the next additional transition must occur. This parameter is written by the CPU and used by the TPU.

The RATIO parameter must be bounded on the upper end to ensure that no normal transition occurs within the interval  $\text{PERIOD} * \text{RATIO}$ . The upper bound is also limited by the requirements that 1) a match on the interval  $\text{PERIOD} * \text{RATIO}$  must occur to identify the detection of an additional transition, and 2) channel service must complete for TCR2 to be reset to \$FFFF. All actions must complete before the next normal transition, which causes TCR2 to increment to \$0000. The range of RATIO is therefore \$00 to \$7F (0 to  $0.99_{10}$ ) for PMA.

#### 4.9 TCR2\_MAX\_VALUE

TCR2\_MAX\_VALUE is a byte containing the maximum permissible value of TCR2. This parameter identifies the case when an additional transition went undetected, and TCR2 did not reset and continued incrementing. If TCR2 exceeds this value due to a normal transition instead of the expected additional transition, TCR2 is set to \$80FF, and ADDITIONAL\_COUNT/BANK\_SIGNAL is cleared to \$0000. This situation is recovered with re-synchronization.

#### 4.10 PERIOD\_HIGH\_WORD

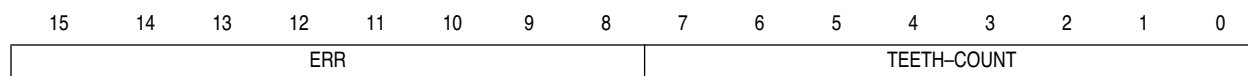
PERIOD\_HIGH\_WORD is the upper eight bits [23:16] of the last measured period. This parameter, with PERIOD\_LOW\_WORD, indicates the TCR1 time duration between the last two input transitions, unless the detected transition is identified as an additional transition. The maximum value in this parameter is \$007E, unless the measured period exceeds \$7EFFFF, in which case this parameter is set to \$007F. PERIOD\_HIGH\_WORD may be read by the CPU at any time and should be read coherently with PERIOD\_LOW\_WORD.

#### 4.11 PERIOD\_LOW\_WORD

PERIOD\_LOW\_WORD is the lower 16 bits [15:0] of the last measured period. This parameter, with PERIOD\_HIGH\_WORD, indicates the TCR1 time duration between the last two input transitions, unless the detected transition is identified as an additional transition. PERIOD\_LOW\_WORD may be read by the CPU at any time and should be read coherently with PERIOD\_HIGH\_WORD.

#### 4.12 TCR2\_VALUE

TCR2\_VALUE is incremented on each transition to track the current value of TCR2. In normal operation (i.e., when no error condition exists), the high byte of this parameter is \$00, and the low byte contains the current value of the low byte of TCR2. This low byte is therefore the current TEETH\_COUNT. When an error condition is detected, the high byte contains an error code and the low byte continues incrementing at each transition detected. The error code parameter is cleared when the error condition disappears. The error codes are shown below in **Figure 5**.



ERR — Error Code	
\$00	No error
\$80	TEETH_COUNT does not match NUM_OF_TEETH on a missing transition detection, or TCR2 exceeds TCR2_MAX_VALUE.
\$C0	(Normal period) or (missing tooth period ÷ 2) exceeds \$FFFF or (PERIOD * RATIO) exceeds \$7FFF (missing tooth interval).

**Figure 5 Error Codes**

## 5 Host Interface to Function

This section provides information concerning the TPU host interface to the function. **Figure 6** 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 6 TPU Address Map**



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

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

CH	Interrupt Enable
0	Channel interrupts disabled
1	Channel interrupts enabled

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

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

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

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

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

CH[15:0]	Action Taken
00	PMA Bank Mode
01	PMA Count Mode
10	(PMM Bank Mode)
11	(PMM Count Mode)

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

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

CH[15:0]	Initialization
00	No Host Service Request
01	Initialization ( <i>Init</i> )
10	Undefined
11	Undefined

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

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

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

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

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

## 6 Function Configuration

The CPU initializes this time function by the following:

1. Writing parameter CHANNEL\_CONTROL:
  - The pin should be configured as input, rising-edge detect, and
  - TCR1 should be used for match and capture;
2. Writing parameters MAX\_ADDITIONAL, TCR2\_MAX\_VALUE, NUM\_OF\_TEETH, and RATIO to parameter RAM;
3. Writing host sequence bits 00 or 01 according to the bank or count mode desired;
4. Issuing an HSR %01 for initialization; and
5. Enabling channel servicing by assigning a high, middle, or low priority.

The TPU then executes initialization. The CPU should monitor the HSR register until the TPU clears the service request to 00 before changing any parameters or before issuing a new service request to this channel. If PMA is used in conjunction with channels executing PSP, the PMA channel should be initialized and enabled at the same time, or before, the PSP channels.

## 7 Performance and Use of Function

### 7.1 Performance

Like all TPU functions, PMA 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. The more TPU channels are active, the more performance decreases. However, worst-case latency in any TPU application can be closely estimated. To analyze the performance of an application that appears to approach the limits of the TPU, use the guidelines given in the TPU reference manual and the information in the following table.

**Table 2 PMA State Timing**

State Name	Clock Cycles	RAM Accesses
S1 <i>Init</i>	16	3
S2 <i>Measure_Period</i>		
No error	58	7
Error	80	10
S3 <i>Additional_Trans</i>		
Normal transitions only	94	8
Additional transitions only — bank mode	40	5
Additional transitions only — count mode	44	6
Additional transitions not detected	8	1

### 7.2 Changing Mode

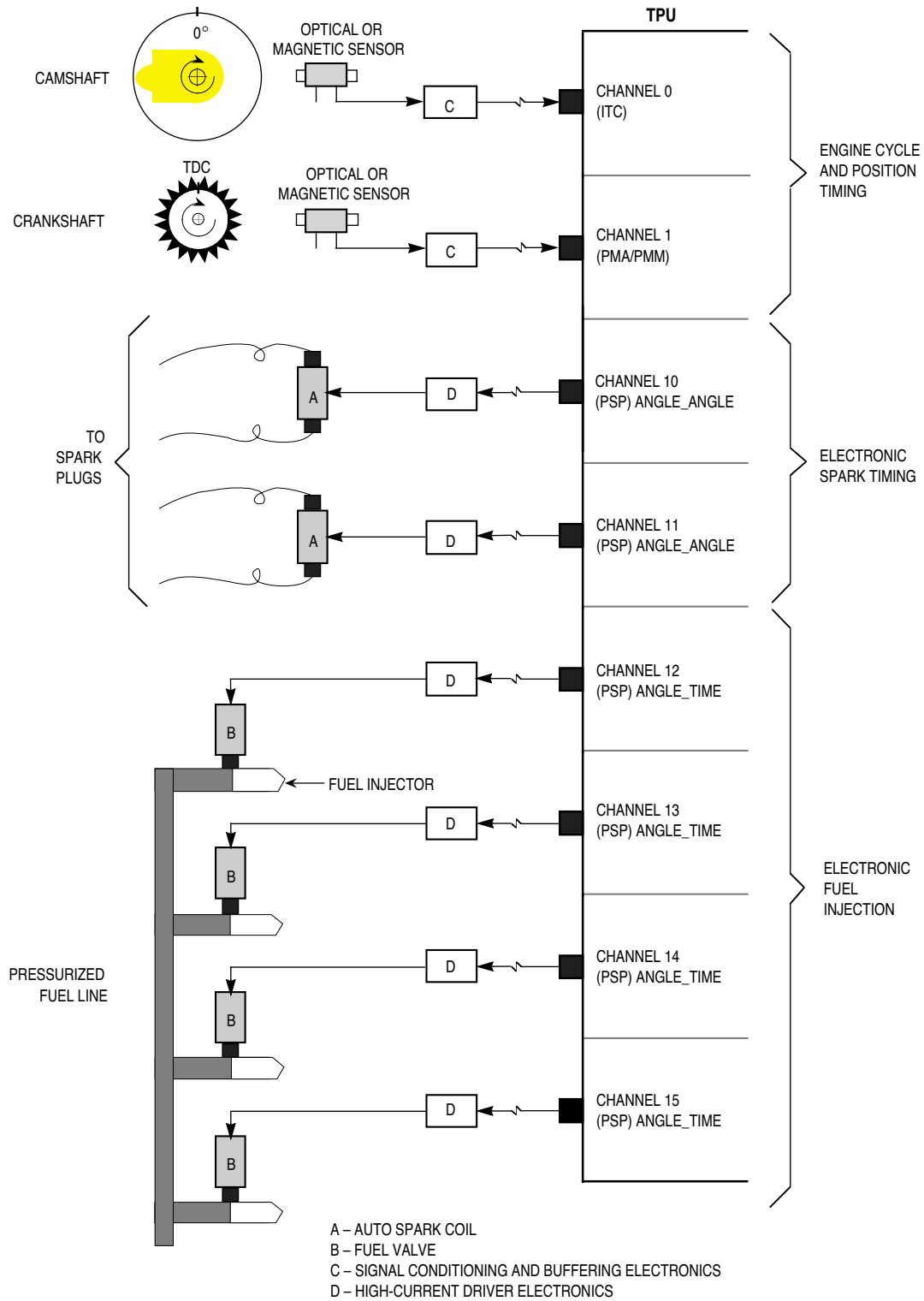
The host sequence bits are used to select PMA function operating mode. Change host sequence bit values only when the function is stopped or disabled (channel priority bits = %00). Disabling the channel before changing mode avoids conditions that cause indeterminate operation.

## 8 Function Examples

### 8.1 Example A

#### 8.1.1 Description

This program demonstrates how to use the ITC, PMA, and PSP functions together to generate an output pulse in relation to an additional tooth. A common application of these functions is an angle-based automotive engine control system. A typical system is shown in **Figure 7**.



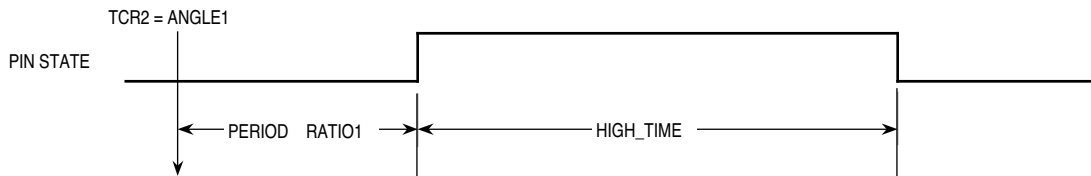
**Figure 7 Engine Control Example A**

**Figure 7** shows that in a typical automotive engine, the camshaft works together with the flywheel to determine the timing for the ignition firing points and fuel-injection pulses. Both the camshaft and the flywheel have reference points in the form of missing or additional teeth. The PMM/PMA functions detect these reference points; the PMM function detects missing teeth, and the PMA function detects additional teeth. This example uses the PMA function to detect additional teeth.

The PMA function has two modes: count mode and bank mode. In count mode, timer TCR2 is reset to \$FFFF after the number of additional transitions in MAX\_ADDITIONAL has been counted. In bank mode, timer TCR2 is reset to \$FFFF after an additional transition has been counted only if BANK\_SIGNAL is a non-zero value. This example uses the bank mode.

In order to use the bank mode, either the CPU or another time function must increment the parameter BANK\_SIGNAL. This example uses the ITC function on another channel to increment BANK\_SIGNAL.

In this example, the PMA function is also used in conjunction with the PSP function. The PMA function determines when the additional tooth occurs, and the PSP function waits a programmable amount of time before it generates an output pulse. The PSP function has two operating modes, angle-angle and angle-time, and it generates the output pulse based on five parameters: RATIO1, RATIO2, ANGLE1, ANGLE2, and HIGH\_TIME. RATIO1 and RATIO2 are 8-bit numbers that represent a decimal multiplier of the period that can range from 0 to 1.99. ANGLE1 and ANGLE2 represent reference angles. A reference angle is simply a tooth number. The teeth are numbered starting with zero after the last additional transition. HIGH\_TIME specifies the time duration of the output pulse in angle-time mode. This example uses the angle-time mode, illustrated in **Figure 8**.



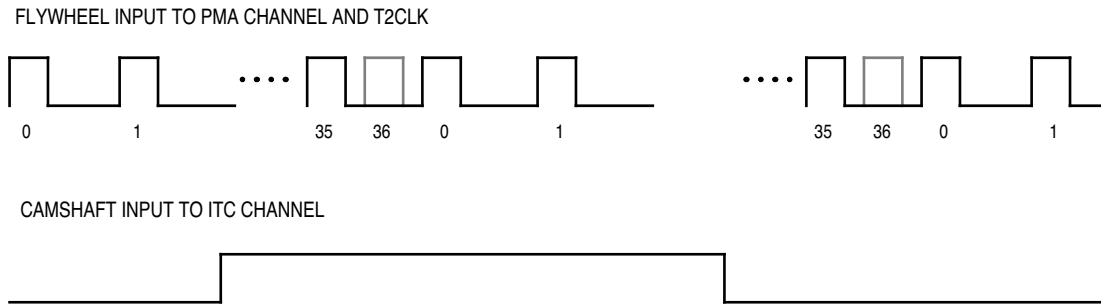
**Figure 8 PSP Angle-Time Mode**

### 8.1.2 Hardware Setup

This example requires two input pulse trains. The input to the TCR2 clock pin and the TPU channel executing the PMA function is a series of pulses with additional transitions. This pulse train is from a flywheel in an automobile engine. In this example, the flywheel has 36 teeth and one additional tooth (37 teeth total). Thus, the teeth are spaced evenly to accommodate 36 teeth, but an additional tooth is placed after the 36th tooth. The flywheel rotates twice for every one rotation of the camshaft. When the flywheel rotates to top dead center and reaches an additional tooth, the engine compresses and fires the spark plugs. After the flywheel makes a second revolution and the camshaft finishes making its first full rotation, the additional tooth is reached again. This time, the engine releases exhaust. The cycle repeats. Thus, the 36-tooth flywheel with one additional tooth behaves the same as a 72-tooth flywheel with two additional teeth. The reference points on the camshaft keep track of which half of the cycle is currently taking place.

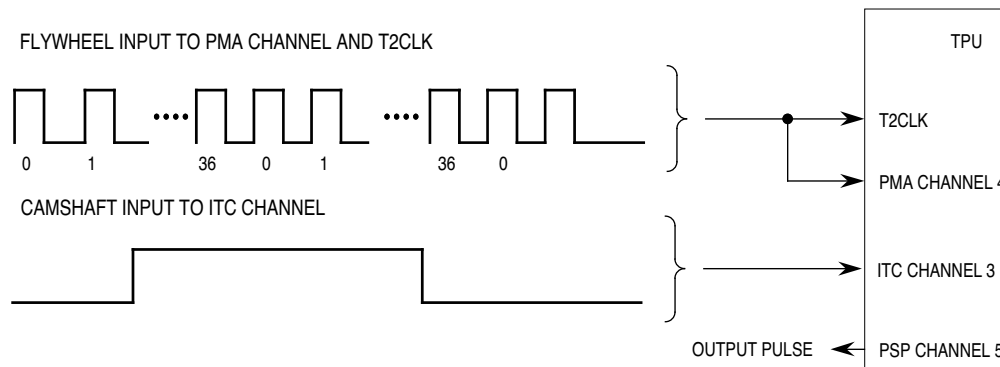
The input pulse train to the channel executing the ITC function consists of high and low transitions from the camshaft. During one additional tooth, this pulse is low, and during the next additional tooth, the pulse is high. This pulse causes BANK\_SIGNAL to increment from zero to one during every other additional tooth.

See **Figure 9** for an illustration of the two input waveforms.



**Figure 9 Input Waveforms for Example A**

The TPU is set up with the functions needed to generate the PSP output pulse: the ITC function on channel 3, the PMA function on channel 4, and the PSP function on channel 5. The ITC function on channel 3 is connected to the camshaft. It is set up so that each time it detects a rising transition, it increments the PMA parameter BANK\_SIGNAL, thus forcing it to a non-zero value to allow timer TCR2 to reset to \$FFFF. The PMA function on channel 4 is connected to the flywheel to detect additional teeth. The PSP function is not physically connected to any of the other channels, but its parameter PERIOD\_ADDRESS points to the PMA parameter PERIOD\_LOW\_WORD. In addition, the TCR2 clock input is connected to the flywheel. See **Figure 10** for an illustration of the hardware setup.



**Figure 10 Example A Hardware Setup**

### 8.1.3 Initialization

Set up channel 3 as ITC counting rising edges from the camshaft. Set up channel 4 as PMA, detecting additional teeth. Set up channel 5 in PSP angle-time mode. The pulse generated will look like the illustration shown on channel 4 in **Figure 1**.

For the ITC channel, the host sequence field bits are 01, continual with no links. The parameter MAX\_COUNT is set to one. On every rising edge, the parameter TRANS\_COUNT will count one edge. When this happens, the TPU will reset TRANS\_COUNT to zero and increment the high byte of the RAM location pointed to by BANK\_ADDRESS. In this case, that byte is BANK\_SIGNAL for the PMA function.

The host sequence field bits for the PMA channel are 00, additional tooth bank mode. When the PMA function operates in the bank mode, TCR2 will not reset to \$FFFF when an additional tooth is reached unless the parameter BANK\_SIGNAL is set to a non-zero value.

## 8.2 Program Code for CPU32-Based Microcontrollers

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

```
TPUMCR equ      $fffe00
TICR   equ      $fffe08
CIER   equ      $fffe0a
CFSR0  equ      $fffe0c
CFSR1  equ      $fffe0e
CFSR2  equ      $fffe10
CFSR3  equ      $fffe12
HSQR0  equ      $fffe14
HSQR1  equ      $fffe16
HSRR0  equ      $fffe18
HSRR1  equ      $fffe1a
CPR0   equ      $fffe1c
CPR1   equ      $fffe1e
CISR   equ      $fffe20
ERROR  equ      $fffffc
PRAM3_0 equ     $ffff30
PRAM3_1 equ     $ffff32
PRAM3_2 equ     $ffff34
PRAM3_3 equ     $ffff36
PRAM3_4 equ     $ffff38
PRAM3_5 equ     $ffff3a
PRAM4_0 equ     $ffff40
PRAM4_1 equ     $ffff42
PRAM4_2 equ     $ffff44
PRAM4_3 equ     $ffff46
PRAM4_4 equ     $ffff48
PRAM4_5 equ     $ffff4a
PRAM5_0 equ     $ffff50
PRAM5_1 equ     $ffff52
PRAM5_2 equ     $ffff54
PRAM5_3 equ     $ffff56
PRAM5_4 equ     $ffff58
PRAM5_5 equ     $ffff5a
        org      $4000           ;begin program at location $4000
        move.w   #$0000,d5       ;d5 initialized to zero
        move.w   #$a000,(CFSR3).l ;Function select field: ITC channel 3, PMA channel 4,
        move.w   #$00cb,(CFSR2).l ;PSP channel 5 (Note: function numbers may vary
        ;for different mask sets)
        move.w   #$0440,(HSQR1).l ;Host Sequence field
        move.w   #$fc0,(CPR1).l  ;Channel priority field: high priority to all channels
```

### 8.2.1 ITC Initialization for Channel 3

```
        move.w   #$0007,(PRAM3_0).l ;Channel control, detect rising edge, use TCR1
        move.w   #$44,(PRAM3_1).l  ;BANK_ADDRESS points to
        ;BANK_SIGNAL
        move.w   #$01,(PRAM3_2).l ;MAX_COUNT = 1
```

### 8.2.2 PMA Initialization for Channel 4

MAX\_ADDITIONAL is a don't care value since bank mode is used.

NUM\_OF\_TEETH is the number that is compared to the current TCR2 count in TCR2\_VALUE when an additional tooth is detected. These numbers must match or else an error code will appear in ERROR. The program NUM\_OF\_TEETH each time an additional tooth is detected. Since the TPU begins counting with zero instead of one, NUM\_OF\_TEETH is first set to 36 instead of 37. NUM\_OF\_TEETH alternates between 36 and 73 (\$24 and \$49) at each additional tooth.

RATIO, multiplied by the previous period, is the upper bound of time in which the next additional transition must occur. This parameter is written by the CPU and used by the TPU.

The `RATIO` parameter must be bounded on the upper end to ensure that no normal transition occurs within the interval `PERIOD * RATIO`. The upper bound is also limited by the requirements that 1) a match on the interval `PERIOD * RATIO` must occur to identify the detection of an additional transition, and

2) channel service must complete for `TCR2` to be reset to `$FFFF`. All actions must complete before the next normal transition, which causes `TCR2` to increment to `$0000`. The range of ratio is therefore `$00` to `$7F` ( $0_{10}$  to  $0.99_{10}$ ) for PMA. In this case, `$60` is used.

`TCR2_MAX_VALUE` contains the maximum permissible value of `TCR2`. `TCR2` resets to `$FFFF` when the final additional tooth is reached so that the tooth count can begin again at zero. If an additional transition goes undetected and `TCR2` continues incrementing past `TCR2_MAX_VALUE`, then the error code `$80` is set in `ERROR`, and `BANK_SIGNAL` is cleared to zero. Since the maximum number of teeth to be counted before the final additional tooth is reached and `TCR2` is reset is 73, set `TCR2_MAX_VALUE` to 72 (`$48`), since the TPU begins numbering with zero.

Since the `NUM_OF_TEETH` parameter alternates between 36 and 73, interrupt the program each time an additional transition is detected. Start the interrupt routine at the label `INT` by storing the address of `INT` in the appropriate vector address location. For this example, the base vector number `$80` is chosen. This number is stored in the `TICR` register. The actual interrupt vector number is calculated by concatenating the channel number with the base vector number. Thus, the interrupt vector number is `$84`, since channel 4 is used. The vector address (where the starting address of the interrupt routine is stored) is calculated as four times the vector number plus the value in the vector base register. In this case, since this program was developed on an M68332 BCC, the vector base register is initialized to `$400` by `CPU32Bug`. The vector address is therefore  $4 * \$84 + \$400$ , which is equal to `$610`.

The interrupt level must be set to a non-zero value in the `TICR`. The interrupt level chosen determines the priority given to this interrupt. Level 7 is the highest priority, and level 1 is the lowest. This example uses level 6. Once an interrupt level has been chosen, bits [10:8] in the CPU status register must be modified to allow recognition of that level interrupt. These bits must be set to a number that is lower than the interrupt level number. Interrupts at the same level or lower than the number in the CPU status register will be masked out and will not be recognized by the CPU. In addition, the interrupt arbitration (`IARB`) field in the `TPUMCR` must be set to a non-zero value between `$0` and `$F`. Otherwise, a spurious interrupt may occur. If two or more interrupts on the same level request an interrupt at the same time, the `IARB` value determines which interrupt will be recognized first.

```

move.w    #$0004,(PRAM4_0).l;Channel control, detect rising edge
move.w    #$0024,(PRAM4_1).l;NUM_OF_TEETH=36
move.w    #$6048,(PRAM4_3).l;RATIO=$60, TCR2_MAX_VALUE = 72
move.w    (CISR).l,d0        ;clear all TPU interrupt requests
move.w    #$0000,(CISR).l
move.l    #INT,($0610).l    ;start interrupt routine at INT
ori.w     #$0005,(TPUMCR).l ;set IARB field
move.w    #$0680,(TICR).l  ;interrupt level 6, base vector=$80
andi.w    #$F5ff,SR        ;allow interrupts on level 6 and above

```

### 8.2.3 PSP Initialization for Channel 5 in Angle-Time Mode

Since the PSP is initialized in angle-time mode, the parameters that form the output pulse are determined as follows:

1. The hightime is specified in `HIGH_TIME`. For this example, it is `$100`.
2. The rising edge is determined by three parameters: `ANGLE1`, `RATIO1`, and `PERIOD_ADDRESS`. `ANGLE1` is a `TCR2` tooth number. Remember that teeth are numbered starting with zero after the additional transition. `PERIOD_ADDRESS` points to the PMA parameter `PERIOD_LOW_WORD`, which contains the period of the input to `TCR2`. `RATIO1` is an 8-bit multiplier that ranges from `$00` to `$FF` (0 to 1.99). The rising edge of the output waveform is offset from `TCR2 = ANGLE1 by PERIOD * RATIO1`.



### 3. The falling edge occurs at the end of HIGH\_TIME.

```
move.w    #$4a01,(PRAM5_0).l ;period address points to period
                                ;low word of PMA, Channel control is a don't care value
move.w    #$b001,(PRAM5_4).l ;RATIO1 = $b0, ANGLE1 = 01
move.w    #$100,(PRAM5_5).l ;HIGH_TIME = $100
start     move.w    #$940,(HSRR1).l ;Host service request for ch 3, 4, and 5
move.w    #$0010,(CIER).l ;enable interrupt for channel 4
finish    bra      finish
```

## 8.2.4 Interrupt Handling Routine for PMA

The processor will be interrupted each time an additional tooth is detected, and the interrupt routine alternates NUM\_OF\_TEETH between 36 (\$24) and 73 (\$49). Because the processor does not know at which tooth it starts counting, it takes a few interrupts (typically about four) to achieve synchronization. Until synchronization is achieved, an error code will appear in ERROR at each interrupt. Thus, until ERROR clears for the first time, keep NUM\_OF\_TEETH at 36. Then, each time an interrupt is received thereafter, alternate NUM\_OF\_TEETH between 36 and 73.

```
INT       andi.w    #$ffef,(CIER).l ;disable interrupt in CIER
move.w    (CISR).l,d6 ;read interrupt
andi.w    #$ffef,(CISR).l ;clear interrupt
cmpi.w    #$01,d5 ;see if synchronization has been achieved
beq       noerr ;if not, change NUM_OF_TEETH
move.w    (ERROR).l,d1 ;check to see if the error bits are clear
cmpi.w    #$0024
beq       noerr ;if they are clear, alternate NUM_OF_TEETH
move.w    #$00ff,(ERROR).l ;if not, clear error bits
move.w    #$0010,(CIER).l ;enable interrupt and return
RTE
noerr     move.w    (PRAM4_1).l,d6 ;if error bits clear, check value
                                ;of NUM_OF_TEETH
cmpi.w    #$0024,d6 ;if NUM_OF_TEETH is 36, change to 73
bne       not36 ;if NUM_OF_TEETH is 73, jump to not36
move.w    #$0049,(PRAM4_1).l
bra       dnch
not36     move.w    #$0024,(PRAM4_1).l ;if NUM_OF_TEETH is 73, change to 36
dnch      move.w    #$0010,(CIER).l ;enable interrupt and return
move.w    #$01,d5 ;flag that synchronization has been reached
RTE
```

## 8.3 Program Code for CPU16-Based Microcontrollers

This program was assembled on the IASM16 Assembler available with the ICD16 In-Circuit Debugger from P&E Microcomputer Systems and was run on an MC68HC16Y1EVb.

```
TPUMCR    equ      $fe00
TICR      equ      $fe08
CIER      equ      $fe0a
CFSR0     equ      $fe0c
CFSR1     equ      $fe0e
CFSR2     equ      $fe10
CFSR3     equ      $fe12
HSQR0     equ      $fe14
HSQR1     equ      $fe16
HSRR0     equ      $fe18
HSRR1     equ      $fe1a
CPR0      equ      $fe1c
CPR1      equ      $fe1e
CISR      equ      $fe20
ERROR     equ      $fffc
PRAM3_0   equ      $ff30
PRAM3_1   equ      $ff32
PRAM3_2   equ      $ff34
PRAM3_3   equ      $ff36
```

```

PRAM3_4      equ      $ff38
PRAM3_5      equ      $ff3a
PRAM4_0      equ      $ff40
PRAM4_1      equ      $ff42
PRAM4_2      equ      $ff44
PRAM4_3      equ      $ff46
PRAM4_4      equ      $ff48
PRAM4_5      equ      $ff4a
PRAM5_0      equ      $ff50
PRAM5_1      equ      $ff52
PRAM5_2      equ      $ff54
PRAM5_3      equ      $fff6
PRAM5_4      equ      $ff58
PRAM5_5      equ      $ff5a

```

### 8.3.1 Initialization

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

```

ORG          $0000          ;put the following reset vector information
                                ;at address $00000 of the memory map
DW           $0000          ;zk=0, sk=0, pk=0
DW           $0200          ;pc=200 -- initial program counter
DW           $3000          ;sp=3000 -- initial stack pointer
DW           $0000          ;iz=0 -- direct page pointer
org          $0400          ;begin program at memory location $0400

```

The following code initializes and configures the system including the software watchdog and system clock. It was written to be used with an EVB.

```

INITSYS:                                ;give initial values for extension registers
                                ;and initialize system clock and COP
LDAB        #$0F
TBEX
LDAB        #$00
TBXK
TBYK
TBZK
TBSK
LDD         #$0003          ;at reset, the CSBOOT block size is 512K.
STD         CSBARBT        ;this line sets the block size to 64K since
                                ;that is what physically comes with the EVB16
LDAA        #$7F
STAA       SYNCR
CLR         SYPCR          ;set system clock to 16.78 MHz
                                ;turn COP (software watchdog) off,
                                ;since COP is on after reset
lds         #$f000
**** MAIN PROGRAM ****
ldab       #$0f
tbek
clr        clrb
tbzk
ldz        #$0000          ;use IZ for indexed offset
clre       ;Accumulator E initialized to zero
ldd        #$a000
std        CFSR3          ;Function select field: ITC channel 3, PMA channel 4,
ldd        #$00cb        ;PSP channel 5 (Note: function numbers may vary
std        CFSR2          ;for different mask sets)
ldd        #$0440
std        HSQR1          ;Host Sequence field
ldd        #$fc0
std        CPR1          ;Channel priority field: high priority to all channels

```

### 8.3.2 ITC Initialization for Channel 3

```

ldd        #$0007

```

```

std      PRAM3_0          ;Channel control, detect rising edge and use TCR1
ldd     #$44
std     PRAM3_1          ;BANK_ADDRESS points to BANK_SIGNAL
ldd     #$01
std     PRAM3_2          ;MAX_COUNT = 1

```

### 8.3.3 PMA Initialization for Channel 4

MAX\_ADDITIONAL is a don't care value since the program uses bank mode.

NUM\_OF\_TEETH is the number that is compared to the current TCR2 count in TCR2\_VALUE when a missing tooth is detected. These numbers must match or else an error code will appear in ERROR. The program changes NUM\_OF\_TEETH at each missing tooth. Since the TPU begins counting with zero instead of one, NUM\_OF\_TEETH is first set to 36 instead of 37.

RATIO, multiplied by the previous period, is the upper bound of time in which the next additional transition must occur. This parameter is written by the CPU and used by the TPU.

The RATIO parameter must be bounded on the upper end to ensure that no normal transition occurs within the interval PERIOD \* RATIO. The upper bound is also limited by the requirements that 1) a match on the interval PERIOD \* RATIO must occur to identify the detection of an additional transition, and 2) channel service must complete for TCR2 to be reset to \$FFFF. All actions must complete before the next normal transition, which causes TCR2 to increment to \$0000. The range of ratio is therefore \$00 to \$7F (0 to 0.99<sub>10</sub>) for PMA. In this case, \$60 is used.

TCR2\_MAX\_VALUE contains the maximum permissible value of TCR2. TCR2 resets to \$FFFF when the last additional tooth is reached so that the count can begin again with zero. If an additional transition goes undetected and TCR2 continues incrementing past TCR2\_MAX\_VALUE then the error code \$80 is set in ERROR, and BANK\_SIGNAL is cleared to zero. Since the maximum number of teeth to be counted before the last tooth is reached and TCR2 is reset is 73, set TCR2\_MAX\_VALUE to 72 (\$48), since the TPU begins numbering with zero.

Since the NUM\_OF\_TEETH parameter alternates between 36 and 73, interrupt the program each time an additional transition is detected. Start the interrupt routine at the label INT by storing the address of INT in the appropriate vector address location. For this example, the base vector number \$80 is chosen. This number is stored in the TICR register. The actual interrupt vector number is calculated by concatenating the channel number with the base vector number. Thus, the interrupt vector number is \$84, since channel 4 is used. The vector address (where the starting address of the interrupt routine is stored) is calculated as two times the vector number. In this case, the vector address is 2 \* \$84, which is equal to \$108.

The interrupt level must be set to a non-zero value in the TICR. The interrupt level chosen determines the priority given to this interrupt. Level 7 is the highest priority, and level 1 is the lowest. This example uses level 6. Once an interrupt level has been chosen, bits [6:4] in the CPU status register must be modified to allow recognition of that level interrupt. These bits must be set to a number that is lower than the interrupt level number. Interrupts at the same level or lower than the number in the CPU status register will be masked out and will not be recognized by the CPU. In addition, the interrupt arbitration (IARB) field in the TPUMCR must be set to a non-zero value between \$0 and \$F. Otherwise, a spurious interrupt may occur. If two or more interrupts on the same level request an interrupt at the same time, the IARB value determines which interrupt will be recognized first.

```

ldd     #$0004
std     PRAM4_0          ;Channel control, detect rising edge
ldd     #$0024
std     PRAM4_1          ;NUM_OF_TEETH=36
ldd     #$6048
std     PRAM4_3          ;RATIO = $a0, TCR2_MAX_VALUE
ldd     CISR             ;clear all TPU interrupt requests
clr     clrd

```

```

std      CISR
ldd      #INT
std      $0108,z      ;start interrupt routine at INT
ldd      TPUMCR
ord      #$0005      ;set IARB field
std      TPUMCR
ldd      #$0680
std      TICR      ;interrupt level 6, base vector=$80
andp     #$ff5f      ;allow interrupts on level 6 and above
                        ;assuming reset values for CCR

```

### 8.3.4 PSP Initialization for Channel 5 in Angle-Time Mode

Since the PSP is initialized in angle-time mode, the parameters that form the output pulse are determined as follows:

1. The hightime is specified in HIGH\_TIME. For this example, it is \$100.
2. The rising edge is determined by three parameters: ANGLE1, RATIO1, and PERIOD\_ADDRESS. ANGLE1 is a TCR2 tooth number. Remember that teeth are numbered starting with zero after the missing transition. PERIOD\_ADDRESS points to the PMA parameter PERIOD\_LOW\_WORD, which contains the period of the input to TCR2. RATIO1 is an 8-bit multiplier that ranges from \$00 to \$FF (0 to 1.99). The rising edge of the output waveform is offset from  $TCR2 = ANGLE1$  by  $PERIOD * RATIO1$ .
3. The falling edge occurs at the end of HIGH\_TIME.

```

ldd      #$4a01
std      PRAM5_0      ;period address points to period low word
                        ;of PMM, Channel control is a don't care value

ldd      #$b001
std      PRAM5_4      ;RATIO1 = $b0, ANGLE1 = 01
ldd      #$100
std      PRAM5_5      ;HIGH_TIME = $100
start    ldd      #$940
std      HSRR1      ;Host service request for ch 3, 4, and 5
ldd      #$0010
std      CIER      ;enable interrupt for channel 4
finish   bra      finish

```

### 8.3.5 Interrupt Handling Routine for PMA

The processor will be interrupted each time an additional tooth is detected, and the interrupt routine alternates NUM\_OF\_TEETH between 36 (\$24) and 73 (\$49). Because the processor does not know at which tooth it starts counting, it takes a few interrupts (typically about four) to achieve synchronization. Until synchronization is achieved, an error code will appear in ERROR at each interrupt. Thus, until ERROR clears for the first time, keep NUM\_OF\_TEETH at 36. Then, each time an interrupt is received thereafter, alternate NUM\_OF\_TEETH between 36 and 73.

```

INT      ldd      CIER
andd     #$ffef
std      CIER      ;disable interrupt in CIER
ldd      CISR      ;read interrupt
andd     #$ffef
std      CISR      ;clear interrupt
tste
bne      noerr     ;see if synchronization has been achieved
ldd      ERROR     ;if not, change NUM_OF_TEETH
cpd     $0024
beq     noerr     ;if they are clear, alternate NUM_OF_TEETH
ldd     $00ff
std     ERROR     ;if not, clear error bits
ldd     $0010
std     CIER      ;enable interrupt and return
RTI
noerr    ldd     PRAM4_1 ;if error bits clear, check value of NUM_OF_TEETH

```

```

        cmpd    #$0024          ;if NUM_OF_TEETH is 36, change to 72
        bne    not36          ;if NUM_OF_TEETH is 72, jump to not36
        ldd    #$0049
        std    PRAM4_1
        bra    dnch
not36   ldd    #$0024
        std    PRAM4_1          ;if NUM_OF_TEETH is 72, change to 36
dnch   ldd    #$0010
        std    CIER            ;enable interrupt and return
        lde    #$01            ;flag that synchronization has been reached
        RTI

```

## 8.4 Example B

### 8.4.1 Description

This example uses the PMA function in the count mode. It produces an output pulse on the PSP channel after every additional tooth. To wait for two or more teeth to pass before producing an output pulse, NUM\_OF\_TEETH must be changed in an interrupt routine such as was done in Example A.

In the count mode, the ITC channel that changes BANK\_ADDRESS between one and zero is not needed. TCR2 will automatically reset to \$FFFF at each additional tooth and generate an output pulse because ADDITIONAL\_COUNT will match MAX\_ADDITIONAL.

### 8.4.2 Software Initialization

Set up channel 4 for the PMA function, detecting additional teeth. Set up channel 5 in PSP angle-time mode. The pulse generated will look like the one shown on channel 1 in **Figure 1**.

The host sequence field bits for the PMA channel are 01, additional tooth count mode. When the PMA function operates in the count mode and MAX\_COUNT is set to one, TCR2 will reset to \$FFFF at each missing tooth. The interrupt routine from Example A is not needed since a PSP output pulse is generated after every additional tooth, and the NUM\_OF\_TEETH remains constant.

### 8.4.3 Hardware Setup

This example requires one input pulse train. The input to the TCR2 clock pin and the TPU channel executing the PMA function is a series of pulses with additional transitions. This pulse train is from a flywheel in an automobile engine. In this example, the flywheel has 36 evenly-spaced teeth and 1 additional tooth (37 teeth total).

See the first waveform in **Figure 9** for an illustration of this one input waveform.

The TPU is set up with the functions needed to generate the PSP output pulse: the PMA function on channel 4 and the PSP function on channel 5. The PMA function on channel 4 is connected to the flywheel to detect additional teeth. The PSP function is not physically connected to any of the other channels, but its parameter PERIOD\_ADDRESS points to the PMA parameter PERIOD\_LOW\_WORD. In addition, the TCR2 clock input is connected to the flywheel. See **Figure 11** for an illustration of the hardware setup.

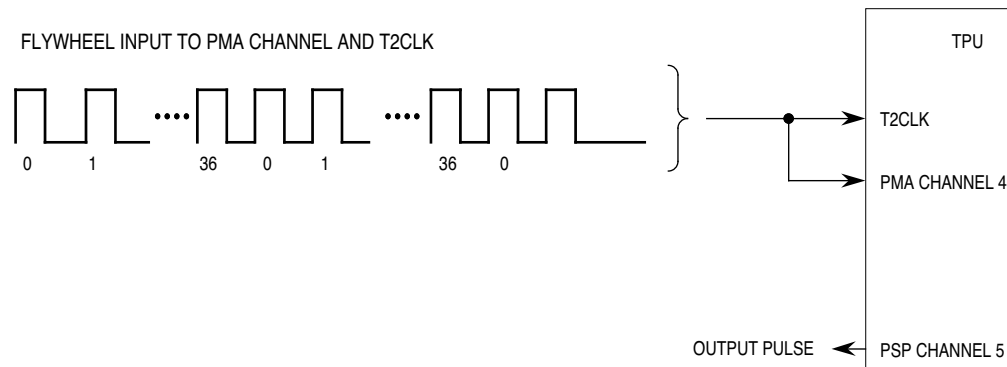


Figure 11 Hardware Setup for Example B

## 8.5 Program Code for CPU32-Based Microcontrollers

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

For this example, use the same equates used for Example A.

Set up channel 4 in PMA count mode, counting additional teeth.

Set up channel 5 in PSP angle-angle mode

```

org          $4000          ;begin program at location $4000
move.w      #$00cb,(CF SR2).l ;function select field
move.w      #$0100,(HSQR1).l ;PSP in angle-angle mode
move.w      #$f00,(CPR1).l  ;high priority to both channels

```

### 8.5.1 PMM Initialization for Channel 4

```

move.w      #$0004,(PRAM4_0).l;Channel Control, detect rising edge
move.w      #$0124,(PRAM4_1).l;MAX_ADDITIONAL = 1, NUM_OF_TEETH = 36
move.w      #$6023,(PRAM4_3).l;RATIO = $60, MAX_VALUE = 35

```

### 8.5.2 PSP Initialization for Channel 5

```

move.w      #$4a01,(PRAM5_0).l;PERIOD_ADDRESS points to
                                     ;PERIOD_LOW_WORD of PMA,
                                     ;Channel Control is a don't care value
move.w      #$3001,(PRAM5_4).l;RATIO1 = $30, ANGLE1 = $01
move.w      #$5002,(PRAM5_5).l;RATIO2 = $50, ANGLE2 = $02
start move.w      #$900,(HSRR1).l ;host service request for channels 4 and 5
finish bra          finish

```

## 8.6 Program Code for CPU16-Based Microcontrollers

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

Set up channel 4 in PMA count mode, counting additional teeth.

Set up channel 5 in PSP angle-angle mode

### 8.6.1 Initialization

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

```

ORG      $0000      ;put the following reset vector information
                        ;at address $00000 of the memory map
DW       $0000      ;zk=0, sk=0, pk=0
DW       $0200      ;pc=200 -- initial program counter
DW       $3000      ;sp=3000 -- initial stack pointer
DW       $0000      ;iz=0 -- direct page pointer
org      $0400      ;begin program at memory location $0400

```

The following code initializes and configures the system including the software watchdog and system clock. It was written to be used with an EVB.

```

INITSYS:                ;give initial values for extension registers
                        ;and initialize system clock and COP

LDAB     #$0F
TBEB     ;point EK to bank F for register access
LDAB     #$00
TBXK     ;point XK to bank 0
TBYK     ;point YK to bank 0
TBZK     ;point ZK to bank 0
TBSK

LDD      #$0003        ;at reset, the CSBOOT block size is 512K.
STD      CSBARBT       ;this line sets the block size to 64K since
                        ;that is what physically comes with the EVB16

LDAA     #$7F          ;w=0, x=1, y=111111
STAA     SYNCR         ;set system clock to 16.78 MHz
CLR      SYPCR         ;turn COP (software watchdog) off,
                        ;since COP is on after reset

lds      #$f000
**** MAIN PROGRAM ****
ldab     #$0f
tbek     ;point to bank $0f for parameter RAM
ldd      #$00cb
std      CFSR2         ;function select field (Note: function numbers may
ldd      #$0100        ;vary for different mask sets)
std      HSQR1         ;PSP in angle-angle mode
ldd      #$f00
std      CPR1         ;high priority to both channels

```

### 8.6.2 PMA Initialization for Channel 4

```

ldd      #$0004
std      PRAM4_0       ;Channel Control, detect rising edge
ldd      #$0124
std      PRAM4_1       ;MAX_ADDITIONAL = 1, NUM_OF_TEETH = 36
ldd      #$6023
std      PRAM4_3       ;RATIO = $60, MAX_VALUE = 35

```

### 8.6.3 PSP Initialization for Channel 5

```

ldd      #$4a01
std      PRAM5_0       ;PERIOD_ADDRESS points to
                        ;PERIOD_LOW_WORD of PMA,
                        ;Channel Control is a don't care value

ldd      #$3001
std      PRAM5_4       ;RATIO1 = $30, ANGLE1 = $01
ldd      #$5002
std      PRAM5_5       ;RATIO2 = $50, ANGLE2 = $02
start    ldd      #$900
std      HSRR1         ;host service request for channels 4 and 5
finish   bra      finish

```

## 9 Function Algorithm

The PMA function consists of three states, which are described in the following paragraphs. The following description is provided as a guide only, to aid understanding of the function. The exact sequence of operations in microcode may be different from that shown, in order to optimize speed and code size.

TPU microcode source listings for all functions in the TPU function library can be downloaded from the Motorola Freeware bulletin board. Refer to *Using the TPU Function Library and TPU Emulation Mode* (TPUPN00/D) for detailed instructions on downloading and compiling microcode.

### 9.1 State 1: *Init*

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 01xxxx.

Match Enable: Don't Care

Summary:

This state is entered as a result of HSR%01. The channel is configured to detect a specified transition at the pin and the particular time base to be used for match and capture events. (In general, TCR1 should be used for both match and capture).

TCR2 and TCR2\_VALUE are set to \$C0FF (since a previous period longer than \$FFFF is assumed). ROLLOVER\_COUNT is set to \$80 (a long previous period is assumed).

Algorithm:

```
Configure channel latches via CHANNEL_CONTROL
Negate flag0
ROLLOVER_COUNT = $80
/* a longer than $FFFF previous period is set */
TCR2 = $C0FF
TCR2_VALUE = $C0FF
ADDITIONAL_COUNT = 0
```

### 9.2 State 2: *Measure\_Period*

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

Match Enable: Don't Care

Summary:

This state is entered as a result of a match or a normal transition detection, when flag0 equals zero. With a match, ROLLOVER\_COUNT is incremented. With a transition, PERIOD is updated, and TCR2\_VALUE low byte (TEETH\_COUNT) is incremented.

If TCR2 exceeds TCR2\_MAX\_VALUE, indicating an error condition, then TCR2 is set to \$80FF and the high byte of TCR2\_VALUE is set to \$80. A search for the additional transition is initiated if the measured period is less than or equal to \$FFFF TCR1 counts; if the period is greater, TCR2 is set to \$C0FF and the high byte of TCR2\_VALUE is set to \$C0. In either error condition, an interrupt is asserted to the CPU.

Algorithm:

```
If MRL = 1 then {
INCR_ROLLOVER: /* address label */
  If ROLLOVER_COUNT - $FF then {
    ROLLOVER_COUNT = ROLLOVER_COUNT + 1
  }
  If (TDL = 0) then {
    If (ROLLOVER_COUNT > 1) then {
      If (TCR2_VALUE(15) = 0) then {
/* issue period error */
        TCR2 = $C0FF
        TCR2_VALUE (high) = $C0
        Assert interrupt request
        Generate a match on ERT + $8000
      }
    }
  }
}
```



```

        Negate MRL                                     /* this match has been handled */
        Negate channel flag0, flag1
        If host sequence bit 0 = 0 then {              /* bank mode */
            BANK_SIGNAL = $00
        }
        Else {                                         /* count mode */
            ADDITIONAL_COUNT = $00
        }
    Else {
        Generate a match on ERT + $8000
        Negate MRL                                     /* this match has been handled */
    }
}
}
If TDL = 1 then {
NEW_PER:                                             /* address label */
                                                    /* see Figure 12. Period Time Calculation. */

    PERIOD_TIME (14:0) = ERT - REF_TIME
    PERIOD_TIME (22:15) = ROLLOVER_COUNT

                                                    /* update period parameters */

    PERIOD_LOW_WORD = PERIOD_TIME [15:0]
    PERIOD_HIGH_WORD = PERIOD_TIME [22:16]           /* upper bits zero */
    REF_TIME = ERT
    ROLLOVER_COUNT = 0
    Negate TDL
    TCR2_VALUE (low) = TCR2_VALUE (low) + 1
    If TCR2 (low) > TCR2_MAX_VALUE then {
        TCR2 = $80FF
        TCR2_VALUE (high) = $80
        Generate a match on ERT + $8000
        Assert interrupt request
        Negate MRL                                     /* this match has been handled */
        Negate channel flag0, flag1
        If host sequence bit 0 = 0 then {              /* bank mode */
            BANK_SIGNAL = $00
        }
        Else {                                         /* count mode */
            ADDITIONAL_COUNT = $00
        }
    }
}
Else {
    If (PERIOD_TIME ≥ $10000) then {                  /* Period error*/

        If (TCR2_VALUE(15) = 0) then {                /* issue period error */

            TCR2 = $C0FF
            TCR2_VALUE(high) = $C0
            Assert interrupt request
            Generate a match on ERT + $8000
            Negate MRL                                 /* this match has been handled */
            Negate channel flag0, flag1
            If host sequence bit 0 = 0 then {          /* bank mode */
                BANK_SIGNAL = $00
            }
            Else {                                     /* count mode */

```



1. In count mode (host sequence bit 0 equals one):
  - If (TCR2 - NUM\_OF\_TEETH), indicating an error condition, then TCR2 is set to \$80FF, the TCR2\_VALUE high byte is set to \$80, the low byte is incremented, and ADDITIONAL\_COUNT is set to \$0000; or else,
  - If (TCR2 = NUM\_OF\_TEETH) then ADDITIONAL\_COUNT is incremented, and if (ADDITIONAL\_COUNT ≥ MAX\_ADDITIONAL), then TCR2 is set to \$FFFF, TCR2\_VALUE is set to \$00FF, and ADDITIONAL\_COUNT is cleared.
2. In bank mode (host sequence bit 0 equals zero):
  - If (TCR2 - NUM\_OF\_TEETH), indicating an error condition, then TCR2 is set to \$80FF, the TCR2\_VALUE high byte is set to \$80, the low byte is incremented, and BANK\_SIGNAL is set to \$00; or else,
  - If (TCR2 = NUM\_OF\_TEETH) and if (BANK\_SIGNAL - \$00) (reference has been passed), then TCR2 is set to \$FFFF, TCR2\_VALUE is set to \$00FF, and BANK\_SIGNAL is set to \$00.

Algorithm:

```

Negate channel flag0
If (TDL = 1) and (MRL = 0) then {                                     /* additional transition detected */
  Negate TDL
  TCR2_VALUE (low) = TCR2_VALUE (low) + 1
  If TCR2 (low) - NUM_OF_TEETH then {
    TCR2 = $80FF
    TCR2_VALUE (high) = $80
    Generate a match on ERT + $8000
    Assert interrupt request
    Negate MRL                                                         /* this match has been handled */
    Negate channel flag0, flag1
    If host sequence bit 0 = 0 then {                                   /* bank mode */
      BANK_SIGNAL = $00
    }
    Else {                                                             /* count mode */
      ADDITIONAL_COUNT = $00
    }
  }
}
Else {
  If (host sequence bit 0 = 0) and (BANK_SIGNAL - $00) then {
    TCR2 = $FFFF
    TCR2_VALUE = $00FF
    BANK_SIGNAL = $00
    Assert interrupt request
  }
  If (host sequence bit 0 = 1) then {
    ADDITIONAL_COUNT = ADDITIONAL_COUNT + 1
    If (ADDITIONAL_COUNT ≥ MAX_ADDITIONAL) then {
      TCR2 = $FFFF
      TCR2_VALUE = $00FF
      ADDITIONAL_COUNT = $00
      Assert interrupt request
    }
  }
}
Prepare a match on REF_TIME + $8000
Negate MRL                                                         /* this match has been handled */
}
If (MRL = 1) and (TDL = 0) then {                                     /* no additional transition detected */
  Prepare a match on REF_TIME + $8000
  Negate MRL                                                         /* this match has been handled */
}

```

```

}
If (MRL = 1) and (TDL = 1) then {                               /* no additional transition detected */
  If (ERT – REF_TIME ≥ $8000) then {
    Goto INCR_ROLLOVER                                           /* calculate new period */
  }
  Goto NEW_PER                                                   /* calculate new period */
}
                                                                    /* EXPLANATION:

```

This state is entered as a result of one of the following conditions:

MRL = 0; TDL = 1:

An additional transition has been detected. Set TCR2 to \$FFFF, prepare a match on REF\_TIME + \$8000, and wait for the next transition and/or rollover period.

MRL = 1; TDL = 0:

No additional transition, prepare a match on REF\_TIME + \$8000, wait for the next transition and/or rollover period.

MRL = 1; TDL = 1:

No additional transition. Calculate new period. Increment ROLLOVER\_COUNT if the last transition time minus REF\_TIME is greater than \$8000. \*/

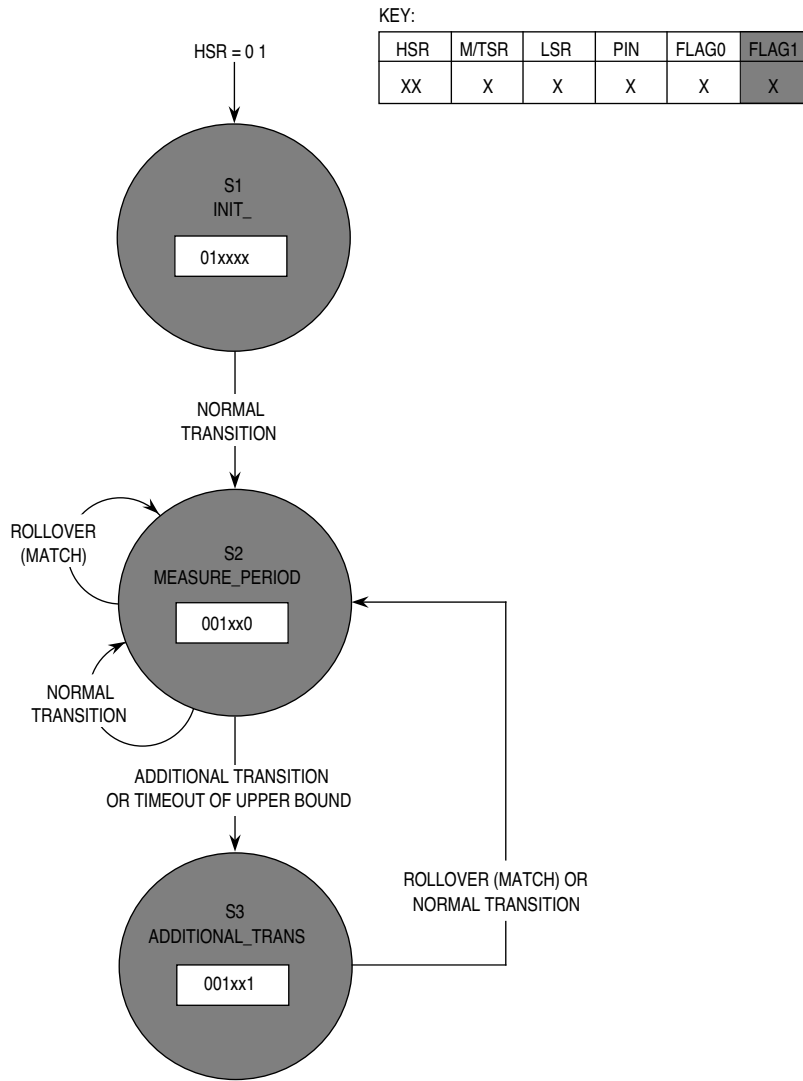
The table below shows the PMA state transitions listing the service request sources and channel conditions from current state to next state. **Figure 13** illustrates the flow of PMA states, including the initialization and immediate update states.

**Table 3 PMA State Transition Table**

Current State	HSR	M/TSR	LSR	Pin	Flag0	Next State
All States	01	—	—	—	—	S1 Init
S1 Init	00	1	—	—	0	S2 Measure_Period
S2 Measure_Period	00	1	—	—	0	S2 Measure_Period
	00	1	—	—	1	S3 Additional_Trans
S3 Additional_Trans	00	1	—	—	0	S2 Measure_Period
	00	1	—	—	1	S3 Additional_Trans
Unimplemented Conditions	11	—	—	—	—	—
	10	—	—	—	—	—
	00	0	1	—	—	—

NOTES:

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



**Figure 13 PMA State Flowchart**

## NOTES

## NOTES

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola 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 Motorola data sheets and/or specifications can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola 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 Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola 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 Motorola was negligent regarding the design or manufacture of the part. MOTOROLA and M are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

**How to reach us:**

**USA/EUROPE/Locations Not Listed:** Motorola Literature Distribution;

P.O. Box 5405, Denver Colorado 80217. 1-800-441-2447, (303) 675-2140

**Mfax™:** RMFAX0@email.sps.mot.com - TOUCHTONE (602) 244-6609, U.S. and Canada Only 1-800-774-1848

**INTERNET:** <http://Design-NET.com>

**JAPAN:** Nippon Motorola Ltd.; Tatsumi-SPD-JLDC,

6F Seibu-Butsuryu-Center, 3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 81-3-3521-8315

**ASIA PACIFIC:** Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park,

51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

Mfax is a trademark of Motorola, Inc.



**MOTOROLA**